

Safety and Reliability for Learning Systems

Part II - A Formal Methods View on Machine Learning
(and a very short introduction to *Linear Programming*)

Rüdiger Ehlers, Clausthal University of Technology

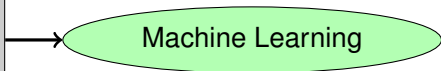
Marktoberdorf Summer School, August 2019

A high-level look at machine learning

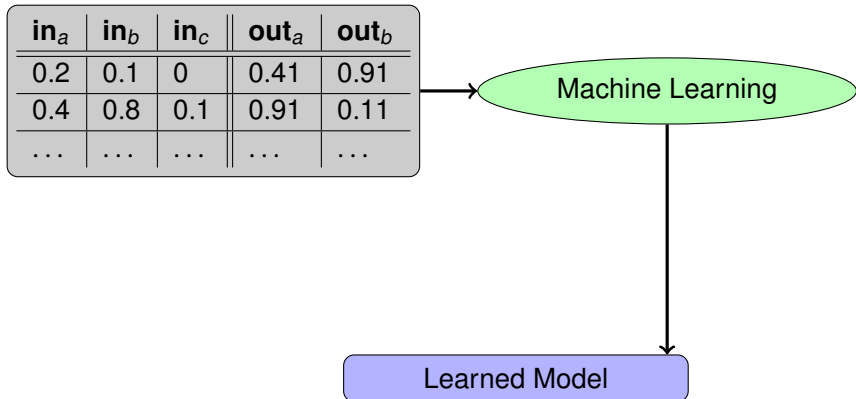
in_a	in_b	in_c	out_a	out_b
0.2	0.1	0	0.41	0.91
0.4	0.8	0.1	0.91	0.11
...

A high-level look at machine learning

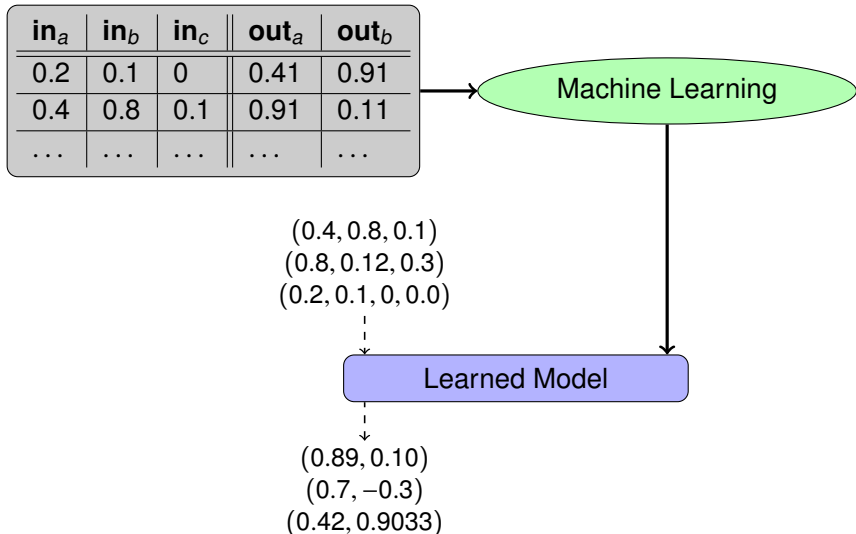
in_a	in_b	in_c	out_a	out_b
0.2	0.1	0	0.41	0.91
0.4	0.8	0.1	0.91	0.11
...



A high-level look at machine learning



A high-level look at machine learning



Overview

So what do we want?

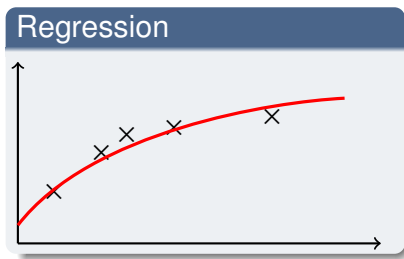
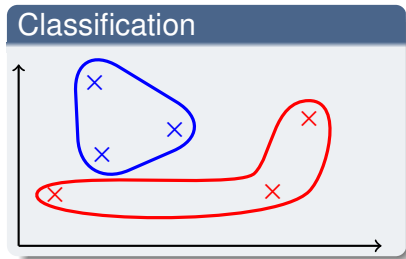
We want some *function/learned model* of some given *type* that fulfils some *soft constraints* (given in the form of *data*).

What is a *machine learning approach*?

A machine learning approach consists of:

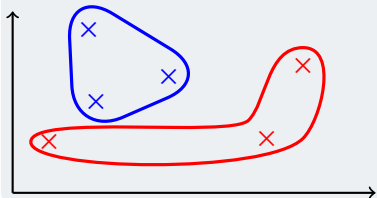
- a *shape* of the learned model and
- a suitable *learning algorithm*.

Classification vs. Regression

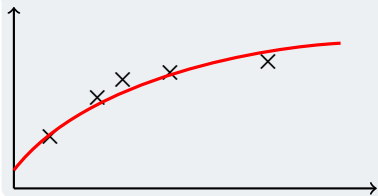


Classification vs. Regression

Classification



Regression

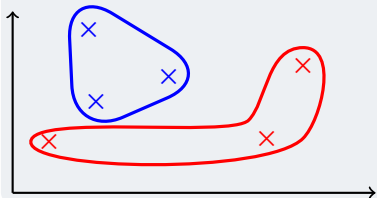


Note

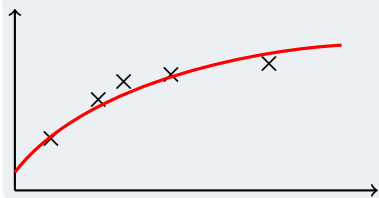
Some machine learning methods are for classification only.

Classification vs. Regression

Classification



Regression



Note

Some machine learning methods are for classification only.

Notes

A machine learning method for regression can *sometimes* be used for classification into $c \in \mathbb{N}$ classes by using:

$$f' = \operatorname{argmax}_{i \in \{1, \dots, c\}} y_i \quad \text{for} \quad (y_1, \dots, y_c) = f(\vec{x})$$

Evaluating machine learning methods

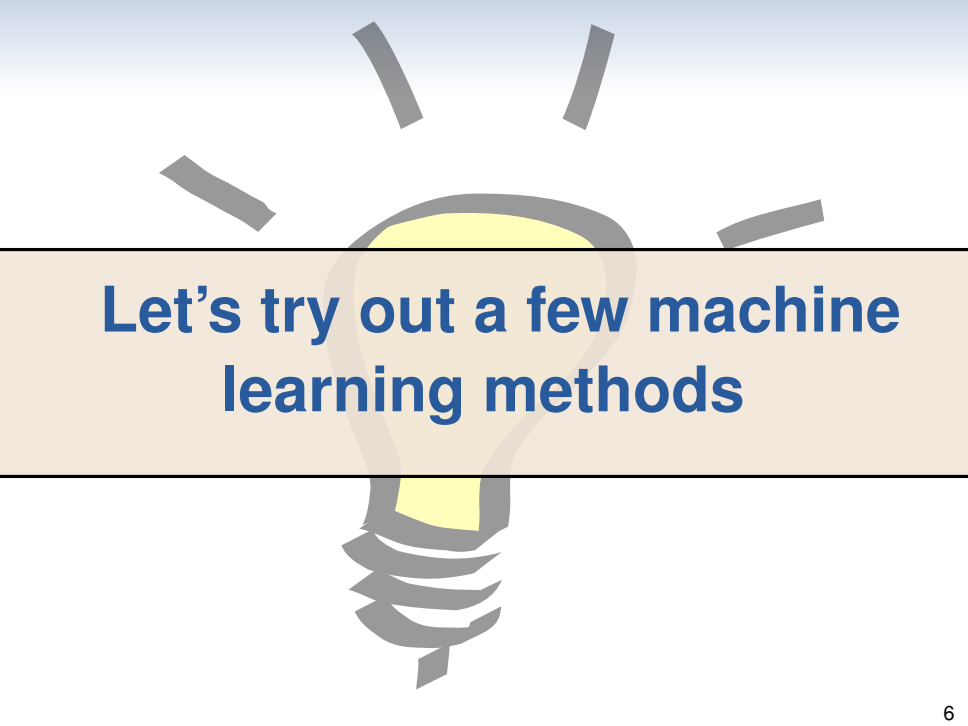
Splitting the data set

The available data is often split into:

- 1 *Training set*
- 2 *Validation set* – to check if learning worked and for selecting a model
- 3 *Testing set* – for evaluating the learned model *at the very end*

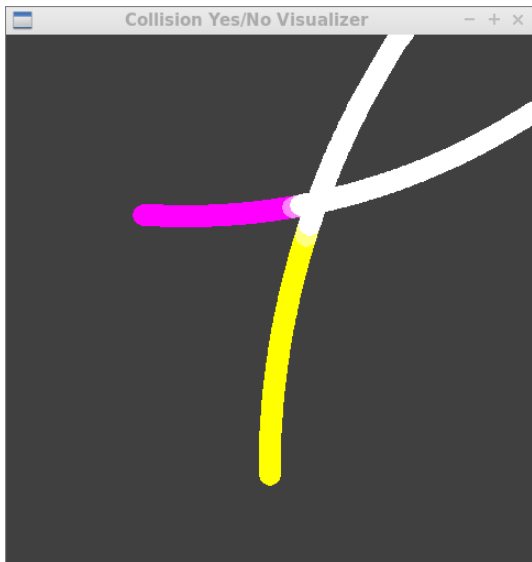
Evaluation metrics (**examples**)

- *Classification*: Accuracy (fraction of data points correctly classified)
- *Regression*: Mean deviation from the output of the data set



**Let's try out a few machine
learning methods**

Collision avoidance benchmark set – Regression



Collision avoidance benchmark set – Regression

Model input

- the relative distances of the vehicles in their workspace in the X- and Y-dimensions,
- the speed of the second vehicle,
- the starting direction of the second vehicle, and
- the rotation speed values of the two vehicles

Model output

Minimum distance between vehicles

Collision avoidance benchmark set – Regression

Rel. X	Rel. Y	v 2nd	obDir	vRot 1st	vRot 2st	minDist
0.09	0.41	0.12	0.58	-0.28	-0.27	129.94
0.33	0.85	0.13	0.47	-0.10	0.02	73.41
0.00	0.44	0.12	0.52	-0.26	-0.15	158.21
0.45	0.91	0.12	0.43	0.17	-0.11	42.68
0.91	0.57	0.15	0.57	0.16	-0.29	149.41
0.80	0.21	0.22	0.51	0.19	-0.09	322.64
0.40	0.77	0.20	0.54	0.08	-0.11	9.26
0.89	0.49	0.23	0.47	-0.28	0.10	250.70
0.18	0.82	0.28	0.53	-0.20	0.15	78.46
0.39	0.07	0.19	0.51	-0.10	-0.10	184.05
0.09	0.63	0.10	0.47	-0.16	-0.19	154.07
0.72	0.91	0.23	0.60	0.28	0.02	113.61
0.23	0.48	0.14	0.60	-0.03	0.08	16.64
...

The basics: Linear regression

First idea

Let's represent the model as a linear function!

Advantage

In this case, we can learn the model using results from linear algebra. Let:

- $d \in \mathbb{N}$ be the number of dimensions,
- X be a $d \times n$ matrix for the input values of the *training set*
- \vec{y} be the output values for the *training set*

We compute:

A model $\vec{m} \in \mathbb{R}^d$ such that

$$\|X\vec{m} - \vec{y}\|_2$$

is minimal.

The basics: Linear regression



First idea

Let's represent the model as a linear function!

Advantage

In this case, we can learn the model using results from linear algebra. Let:

- $d \in \mathbb{N}$ be the number of dimensions,
- X be a $d \times n$ matrix for the input values of the *training set*
- \vec{y} be the output values for the *training set*

We compute:

A model $\vec{m} \in \mathbb{R}^d$ such that

$$\|X\vec{m} - \vec{y}\|_2$$

is minimal.

The Basics: Regression with add'l basis fn's (1)

Problem with linear regression

The computed model is linear, whereas the underlying real-world phenomenon may not be linear.

The Basics: Regression with add'l basis fn's (1)

Problem with linear regression

The computed model is linear, whereas the underlying real-world phenomenon may not be linear.

Candidate solution

We can learn a linear combination over a set of *basis functions* instead.

The Basics: Regression with add'l basis fn's (1)

Problem with linear regression

The computed model is linear, whereas the underlying real-world phenomenon may not be linear.

Candidate solution

We can learn a linear combination over a set of *basis functions* instead.

Example basis functions

Assuming that the input to the model is a tuple (x_1, \dots, x_6) :

- x_1
- $x_2 \cdot x_3$
- $\sin(x_1)$
- ...

The Basics: Regression with add'l basis fn's (2)

Why work with basis functions?

- We can augment every input vector with the corresponding basis function values
- The model learning problem is then still linear \rightarrow same computation as before.

The Basics: Regression with add'l basis fn's (2)

Why work with basis functions?

- We can augment every input vector with the corresponding basis function values
- The model learning problem is then still linear \rightarrow same computation as before.

Disadvantages

- Using the usual computation approach, the complexity of the approach is:
 - **cubic** in the number of factors
 - **linear** in the number of data points in the training set
- The basis functions need to be chosen well.

The Basics: Regression with add'l basis fn's (A)



Why work with basis functions?

- We can augment every input vector with the corresponding basis function values
- The model learning problem is then still linear \rightarrow same computation as before.

Disadvantages

- Using the usual computation approach, the complexity of the approach is:
 - **cubic** in the number of factors
 - **linear** in the number of data points in the training set
- The basis functions need to be chosen well.

Classification with a linear separator

Starting point

Now we only want to classify whether the cars crash or not.

Classification with a linear separator

Starting point

Now we only want to classify whether the cars crash or not.

Idea

Let us learn a linear classifier between the crashing and non-crashing cases.

Coming up next: a concrete example

Linear separators – an example (1)

Sample data

Let us assume that we only have 2 input dimensions and four (training) data points:

- $x_1 = 0.3, x_2 = 0.8 \rightarrow$ no crash
- $x_1 = 0.1, x_2 = 0.9 \rightarrow$ crash
- $x_1 = 0.7, x_2 = 0.5 \rightarrow$ no crash
- $x_1 = 0.05, x_2 = 0.2 \rightarrow$ crash

Linear separators – an example (1)

Sample data

Let us assume that we only have 2 input dimensions and four (training) data points:

- $x_1 = 0.3, x_2 = 0.8 \rightarrow$ no crash
- $x_1 = 0.1, x_2 = 0.9 \rightarrow$ crash
- $x_1 = 0.7, x_2 = 0.5 \rightarrow$ no crash
- $x_1 = 0.05, x_2 = 0.2 \rightarrow$ crash

Model shape

We use the method for computing a classifier from slide 4.
So we have four parameters $w_1 \dots w_n$ to be learned:

$$w_1 \cdot x_1 + w_2 \cdot x_2 = \text{“likelihood” of crash}$$

$$w_3 \cdot x_1 + w_4 \cdot x_2 = \text{“likelihood” of no crash}$$

Linear separators – an example (2)

Sample data

- $x_1 = 0.3, x_2 = 0.8 \rightarrow$ no crash
- $x_1 = 0.1, x_2 = 0.9 \rightarrow$ crash
- $x_1 = 0.7, x_2 = 0.5 \rightarrow$ no crash
- $x_1 = 0.05, x_2 = 0.2 \rightarrow$ crash

Constraints

$$w_1 \cdot 0.3 + w_2 \cdot 0.8 < w_3 \cdot 0.3 + w_4 \cdot 0.8$$

$$w_1 \cdot 0.1 + w_2 \cdot 0.9 > w_3 \cdot 0.1 + w_4 \cdot 0.9$$

$$w_1 \cdot 0.7 + w_2 \cdot 0.5 < w_3 \cdot 0.7 + w_4 \cdot 0.5$$

$$w_1 \cdot 0.05 + w_2 \cdot 0.2 > w_3 \cdot 0.9 + w_4 \cdot 0.2$$

Linear separators – an example (2)

Sample data

- $x_1 = 0.3, x_2 = 0.8 \rightarrow$ no crash
- $x_1 = 0.1, x_2 = 0.9 \rightarrow$ crash
- $x_1 = 0.7, x_2 = 0.5 \rightarrow$ no crash
- $x_1 = 0.05, x_2 = 0.2 \rightarrow$ crash

Constraints

$$(w_1 - w_3) \cdot 0.3 + (w_2 - w_4) \cdot 0.8 < 0$$

$$(w_1 - w_3) \cdot 0.1 + (w_2 - w_4) \cdot 0.9 > 0$$

$$(w_1 - w_3) \cdot 0.7 + (w_2 - w_4) \cdot 0.5 < 0$$

$$(w_1 - w_3) \cdot 0.05 + (w_2 - w_4) \cdot 0.2 > 0$$

Linear separators – an example (2)

Sample data

- $x_1 = 0.3, x_2 = 0.8 \rightarrow$ no crash
- $x_1 = 0.1, x_2 = 0.9 \rightarrow$ crash
- $x_1 = 0.7, x_2 = 0.5 \rightarrow$ no crash
- $x_1 = 0.05, x_2 = 0.2 \rightarrow$ crash

Constraints

$$w'_1 \cdot 0.3 + w'_2 \cdot 0.8 < 0$$

$$w'_1 \cdot 0.1 + w'_2 \cdot 0.9 > 0$$

$$w'_1 \cdot 0.7 + w'_2 \cdot 0.5 < 0$$

$$w'_1 \cdot 0.05 + w'_2 \cdot 0.2 > 0$$



Begin: Excursion to Linear Programming

Linear Programming – Definition

Setup

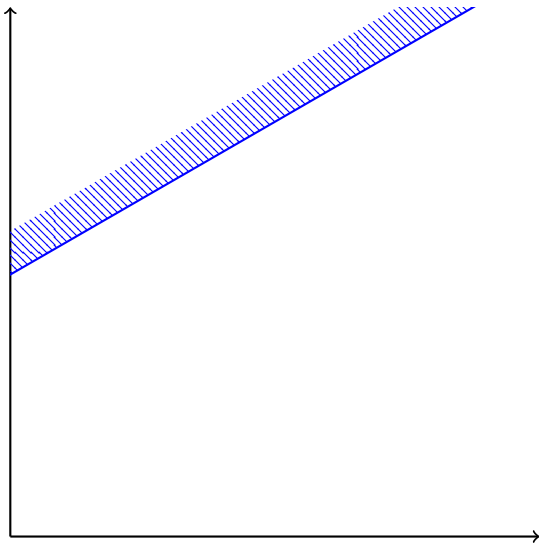
Let

- \mathcal{V} be a set of (positive) real-valued variables,
- C be a set of linear equalities and inequalities over \mathcal{V} , and
- f be a linear function over \mathcal{V} .

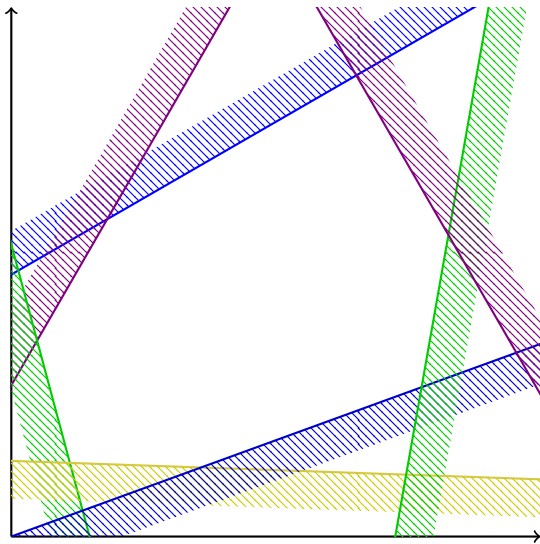
Problem

Find a valuation to \mathcal{V} that satisfies all constraints from C and that maximizes f .

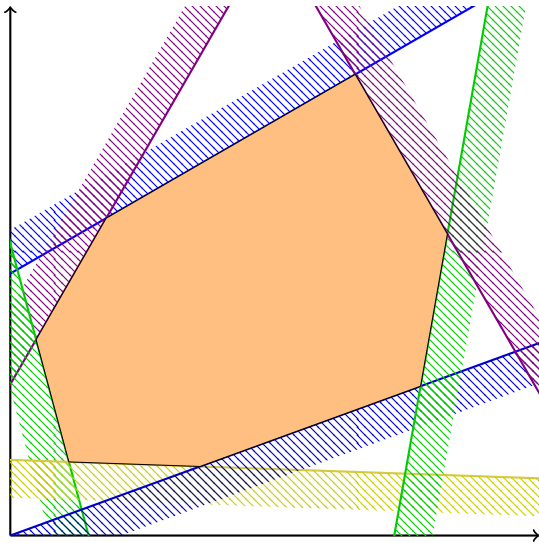
Geometric interpretation of a 2-dimensional LP problem



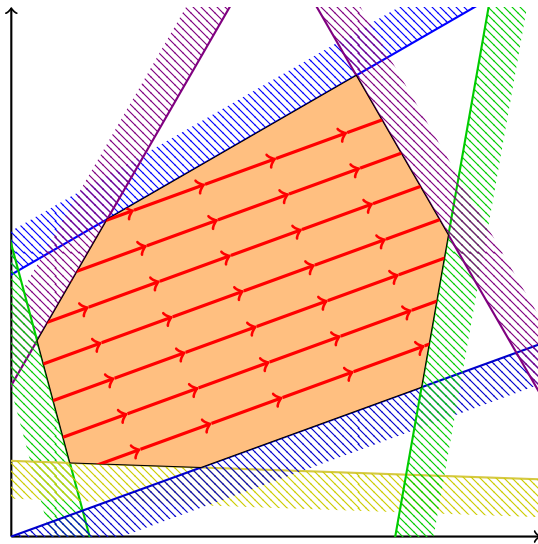
Geometric interpretation of a 2-dimensional LP problem



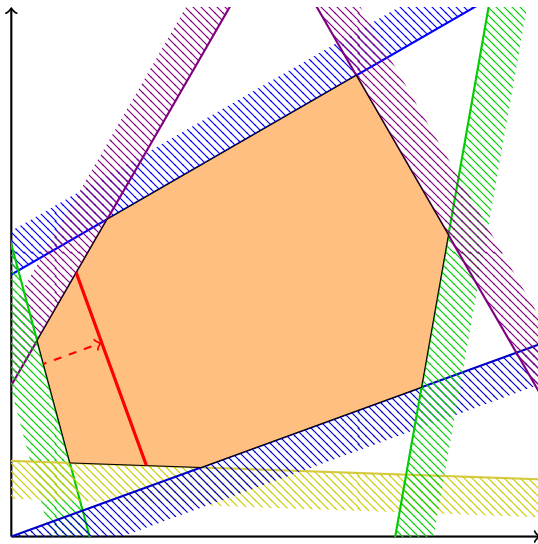
Geometric interpretation of a 2-dimensional LP problem



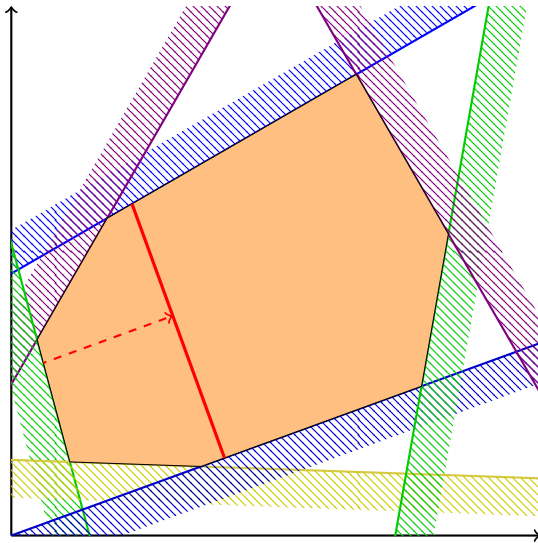
Geometric interpretation of a 2-dimensional LP problem



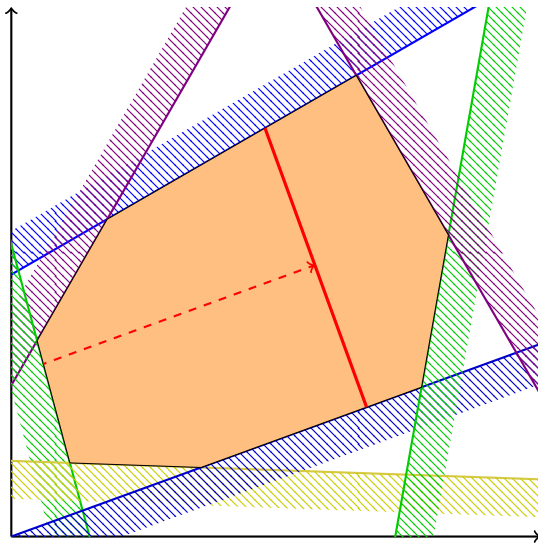
Geometric interpretation of a 2-dimensional LP problem



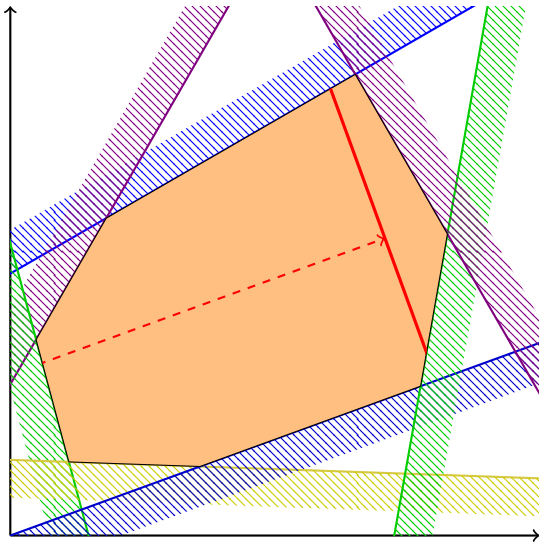
Geometric interpretation of a 2-dimensional LP problem



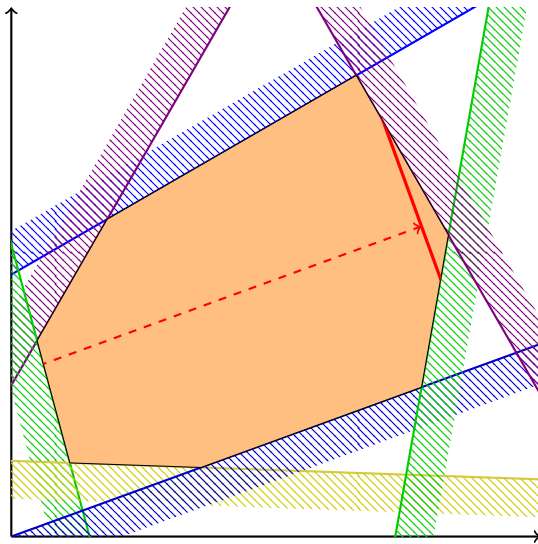
Geometric interpretation of a 2-dimensional LP problem



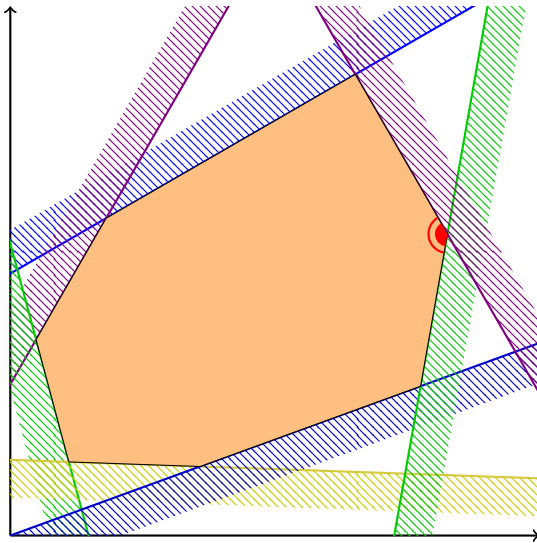
Geometric interpretation of a 2-dimensional LP problem



Geometric interpretation of a 2-dimensional LP problem



Geometric interpretation of a 2-dimensional LP problem



Algorithms for Linear programming

The simplex algorithm (used in practice)

- Idea: Starting from a corner of the solution space, follow the edges of the solution space until no better solution is found.
- Complexity: Exponential in the number of dimensions

Better (in theory): Karmarkar's algorithm

- Introduced by Narendra K. Karmarkar in 1984
- *“In the worst case, the algorithm requires $O(n^{3.5}L)$ arithmetic operations on $O(L)$ bit numbers, where n is the number of variables and L is the number of bits in the input.”*



End: Excursion to Linear Programming

Coming back to our example (1)



Constraints

$$w'_1 \cdot 0.3 + w'_2 \cdot 0.8 < 0$$

$$w'_1 \cdot 0.1 + w'_2 \cdot 0.9 > 0$$

$$w'_1 \cdot 0.7 + w'_2 \cdot 0.5 < 0$$

$$w'_1 \cdot 0.8 + w'_2 \cdot 0.2 > 0$$

Encoding

min: x1;

0.3*x1+0.8*x2 <= -0.1;

0.1*x1+0.9*x2 >= 0.1;

0.7*x1+0.5*x2 <= -0.1;

0.05*x1+0.2*x2 >= 0.1;

free x1, x2;

Coming back to our example (2)



Constraints

$$w'_1 \cdot 0.3 + w'_2 \cdot 0.8 < 0$$

$$w'_1 \cdot 0.1 + w'_2 \cdot 0.9 > 0$$

$$w'_1 \cdot 0.7 + w'_2 \cdot 0.5 < 0$$

$$w'_1 \cdot 0.8 + w'_2 \cdot 0.2 > 0$$

Encoding

min: x1;

$$0.3 * x1 + 0.8 * x2 \leq -0.1;$$

$$0.1 * x1 + 0.9 * x2 \geq 0.1;$$

$$0.7 * x1 + 0.5 * x2 \leq -0.1;$$


$$0.05 * x1 + 0.2 * x2 \geq 0.1;$$

$$-100 \leq x1 \leq 100;$$











$$-100 \leq x2 \leq 100;$$

Applying to our case study



Jupyter LPSupportVectorClassification Last Checkpoint: 07/20/2019 (autosaved)  Logout

File Edit View Insert Cell Kernel Help Not Trusted Python 3

      Run    Markdown 

Example for classification using an LP solver

In [1]:

```
1 # Import libraries
2 import numpy
```

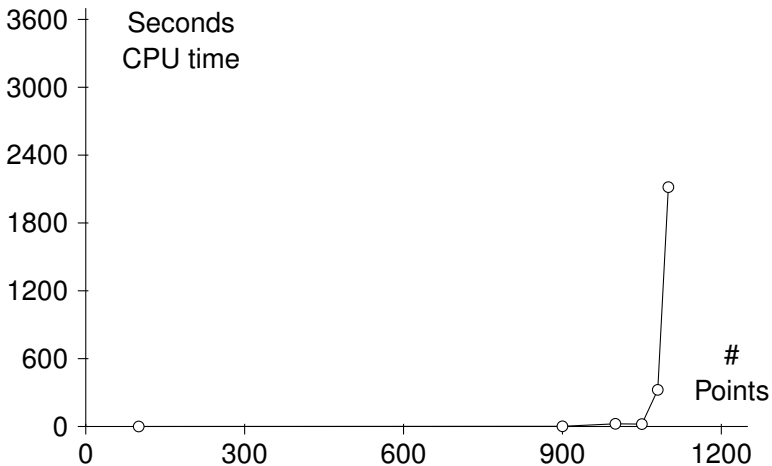
1. Load input data

In [2]:

```
1 # Read CSV file
2 with open("collisions_regression.csv","r") as inFile:
3     allCSVLines = [a.strip().split(",") for a in inFile.readlines()]
4 allCSVLines = [[float(a) for a in b] for b in allCSVLines]
5
6 # Visualize a few lines of the output
7 from IPython.display import HTML, display
8 import tabulate
9 table = allCSVLines[0:9]
10 display(HTML(tabulate.tabulate(table, tablefmt='html')))
```

0.0871867	0.407242	0.12154	0.58024	-0.277108	-0.268582	129.944
0.332198	0.852087	0.131932	0.467443	-0.0997222	0.0217212	73.4099

Computation time with a linear programming tool



Successive approximation for regression

Alternative

We start with some (random) “solution” and successively “push” it into a direction that makes it better

Successive approximation for regression

Alternative

We start with some (random) “solution” and successively “push” it into a direction that makes it better

- Newton’s gradient descent approach (or a variant of it)
- Can get stuck in local optimal, but can deal with lots of data.
- The basis for most modern machine learning approaches

Successive approximation for regression

Alternative

We start with some (random) “solution” and successively “push” it into a direction that makes it better

- Newton’s gradient descent approach (or a variant of it)
- Can get stuck in local optimal, but can deal with lots of data.
- The basis for most modern machine learning approaches

Basic idea

With modern machine learning frameworks such as pytorch, it suffices to

- 1 define variables/matrices as mutable
- 2 define a **loss** function (that is to be minimized)
- 3 invoke an **optimizer** to minimize the loss function

Successive approximation for regression



Alternative

We start with some (random) “solution” and successively “push” it into a direction that makes it better

- Newton’s gradient descent approach (or a variant of it)
- Can get stuck in local optimal, but can deal with lots of data.
- The basis for most modern machine learning approaches

Basic idea

With modern machine learning frameworks such as pytorch, it suffices to

- 1 define variables/matrices as mutable
- 2 define a **loss** function (that is to be minimized)
- 3 invoke an **optimizer** to minimize the loss function

Successive approximation for classification

Implementing classification

Use *trick* from before:

- Use a two-dimensional model output
- Treat the output number with the highest value as the classification result
- Use a loss function that penalizes wrong or close classifications

Successive approximation for classification



Implementing classification

Use *trick* from before:

- Use a two-dimensional model output
- Treat the output number with the highest value as the classification result
- Use a loss function that penalizes wrong or close classifications

Artificial Neural Networks

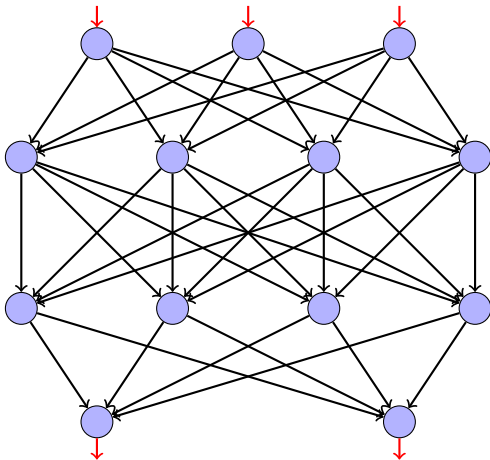
Problem with adding basis functions

- We need to select basis functions by hand
- Using too many basis functions often leads to overfitting
- We need a way to figure out the necessary *features* of the network during learning automatically

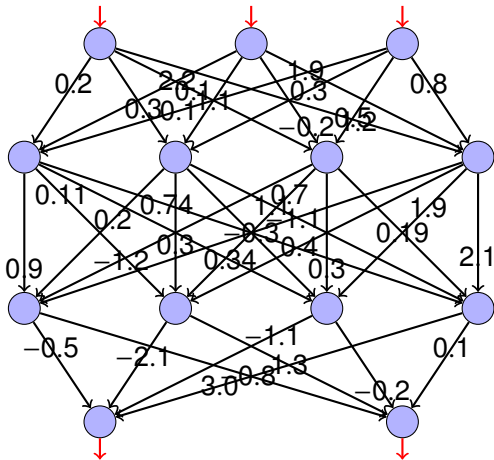
Artificial neural networks

- Permit building arbitrarily complex functions by stitching together *linear layers* with *non-linear activation functions*
- Well-established model in practice
- Permits successive approximation (but can get stuck in local optima)

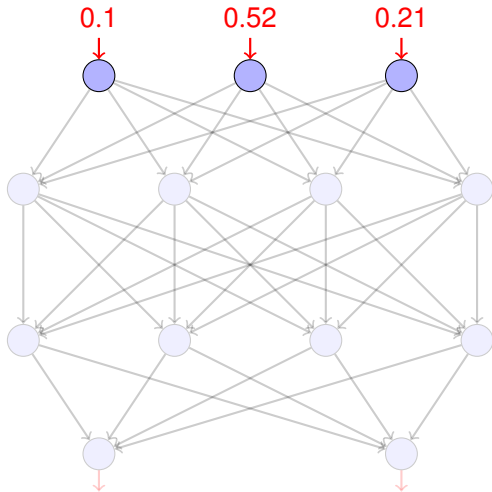
How do FF-NNs work?



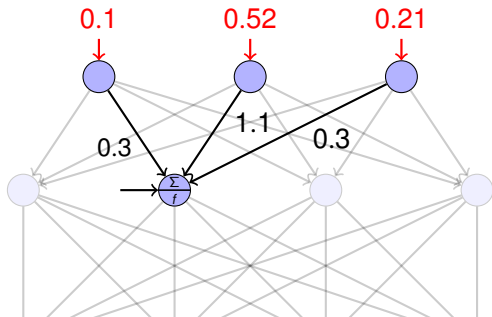
How do FF-NNs work?



How do FF-NNs work?



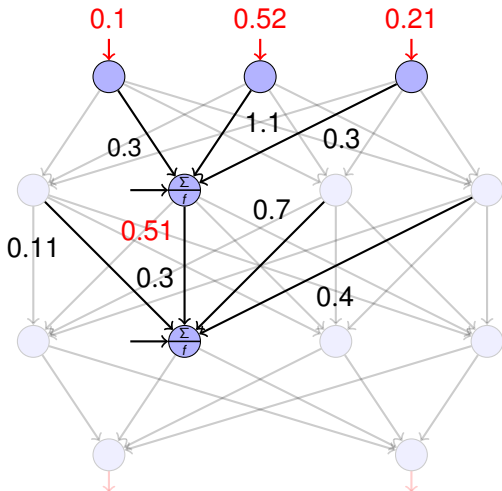
How do FF-NNs work?



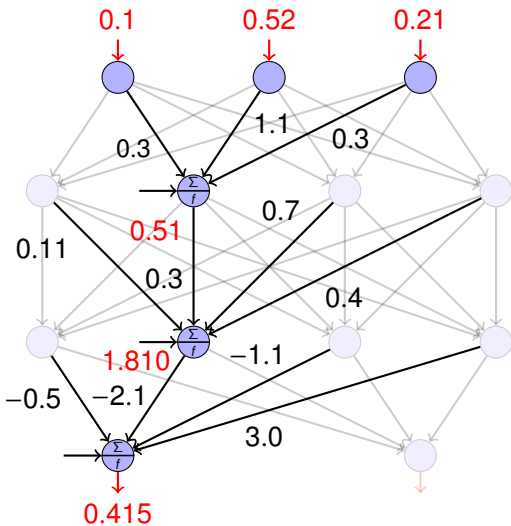
Computing the output of a node

- 1 Compute weighted sum of the incoming flows
- 2 Apply activation function f (e.g., \tanh)

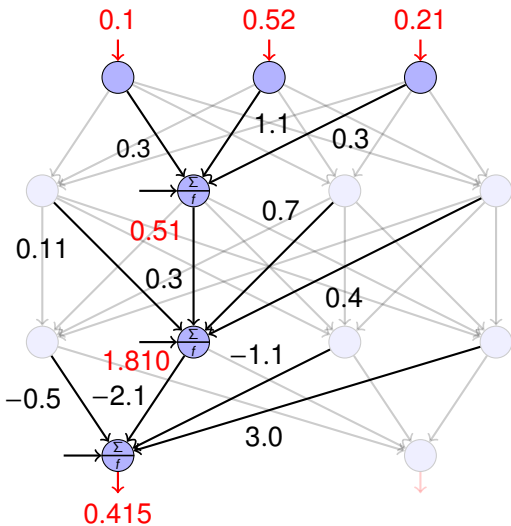
How do FF-NNs work?



How do FF-NNs work?



How do FF-NNs work?



Model shape selection for neural networks

Important considerations

- Number of layers
- Type of the non-linear activation function
- Loss function
- Width of the layers
- ...

Topics not covered

Basics

- Most evaluation metrics for learned systems
- Standard loss functions

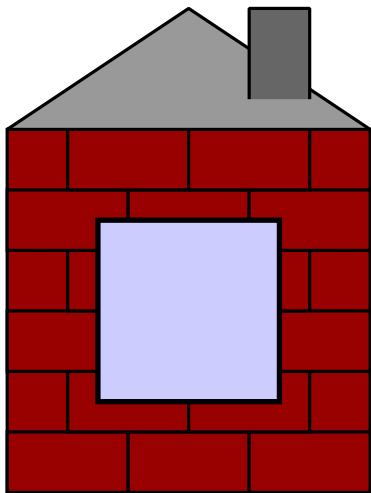
Model types

- Support Vector Machines
- Models with *memory* (e.g., recurrent neural networks)

Engineering

When to choose which method? How do we set the hyperparameters?

Take-home observations for formal verification



- Linear programming allows to reason over continuous functions
- Non-linear activation functions or basis functions are necessary for good learning