PROBABILISTIC AND DIFFERENTIABLE PROGRAMMING

V1: INTRODUCTION

Özgür L. Özçep

Universität zu Lübeck

Institut für Informationssysteme

What this course is about

Differentiable Programming and
Probabilistic Programming for
Machine Learning[1)]

1) Yes, this is a footnote on a slide, believe it or not. The three lines summarizing the topic of the course is the optimal outcome w.r.t my subjective measure - using a non-gradient optimziation procedure starting from the original course name: Probabilistic Differential Programming -> Probabilistic and Differential Programming -> Probabilistic and Differentiable Programming ->Differentiable and Probabilistic Programming

What this lecture V1 is about

Agenda

2. Differentiable Programming and
3. Probabilistic Programming for
1. Machine Learning[1)]

Pointers to lectures in this fancy format.

1) Yes, this is a footnote on a slide, believe it or not. The three lines summarizing the topic of the course is the optimal outcome w.r.t my subjective measure - using a non-gradient optimziation procedure starting from the original course name: Probabilistic Differential Programming -> Probabilistic and Differential Programming -> Probabilistic and Differentiable Programming ->Differentiable and Probabilistic Programming

MACHINE LEARNING

What We Mean by “Learning”

Machine learning (ML) is programming algorithms for

- optimizing a performance criterion
- using example (training) data
- by constructing general(izable) models
- that are good approximations of the data

Role of Mathematics

- Building mathematical model
- core task is inference from a sample

Role of CS: Efficient algorithms

- solve the optimization problem
- represent and evaluate the model for inference

Differentiable Programming

Machine learning (ML) is programming algorithms for

- optimizing a performance criterion
- using example (training) data
- by constructing general(izable) models
- that are good approximations of the data

Role of Mathematics

- Programming differentiable model
- core task is inference from a sample

Role of CS: Efficient algorithms

- solve the optimization problem
- represent and evaluate the model for inference

Probabilistic Programming

Machine learning (ML) is programming algorithms for

- optimizing a performance criterion
- using example (training) data
- by constructing general(izable) models
- that are good approximations of the data

Role of Mathematics

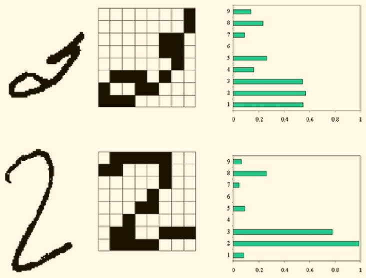
- Programming probabilistic model
- core task is inference from a sample

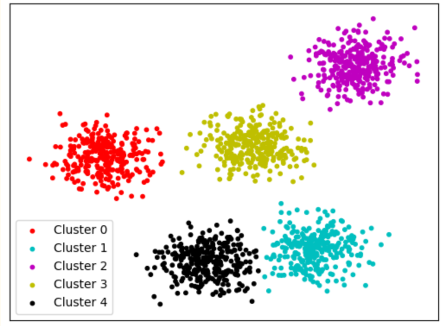
Role of CS: Efficient algorithms

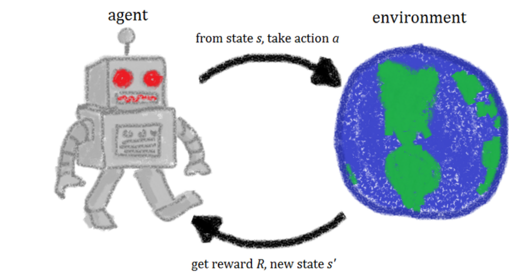
- solve the optimization problem
- represent and evaluate the model for inference

Types of learning (classically)

- Supervised Learning
  learn to predict an output for input vector after training with labelled data
- Unsupervised Learning
  discover a good internal representation of the input
- Reinforcement Learning
  learn to select an action to maximize the expectation of future rewards (payoff)



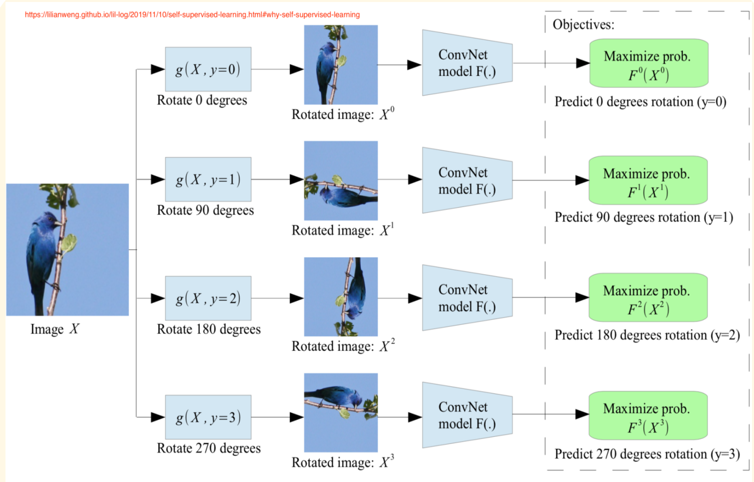




Reference: Wikipedia
https://simple.wikipedia.org/wiki/Reinforcement_learning

Subtypes of unsupervised l. (in Deep Learning context)

- Self-supervised (Self-taught) Learning - learn with targets induced by a prior on the unlabelled training data
- Semi-supervised Learning learn with few labelled examples and many unlabelled ones (same distribution for labelled & non-labelled data)



Generative vs. Discriminative/descriptive

- Many unsupervised and self-supervised models can be classed as ‘generative models’.
  - Given unlabelled data X, a unsupervised generative model learns full joint probability distribution P(X,Y).
  - These are characterised by an ability to ‘sample’ the model to ‘create’ new data
- In contrast: Discriminative models learn P(Y|X) (which can be calculated in a generative model, too, using Bayes‘s rule but not vice versa)

(X: observations, data, Y: categories, classes, non-observed)

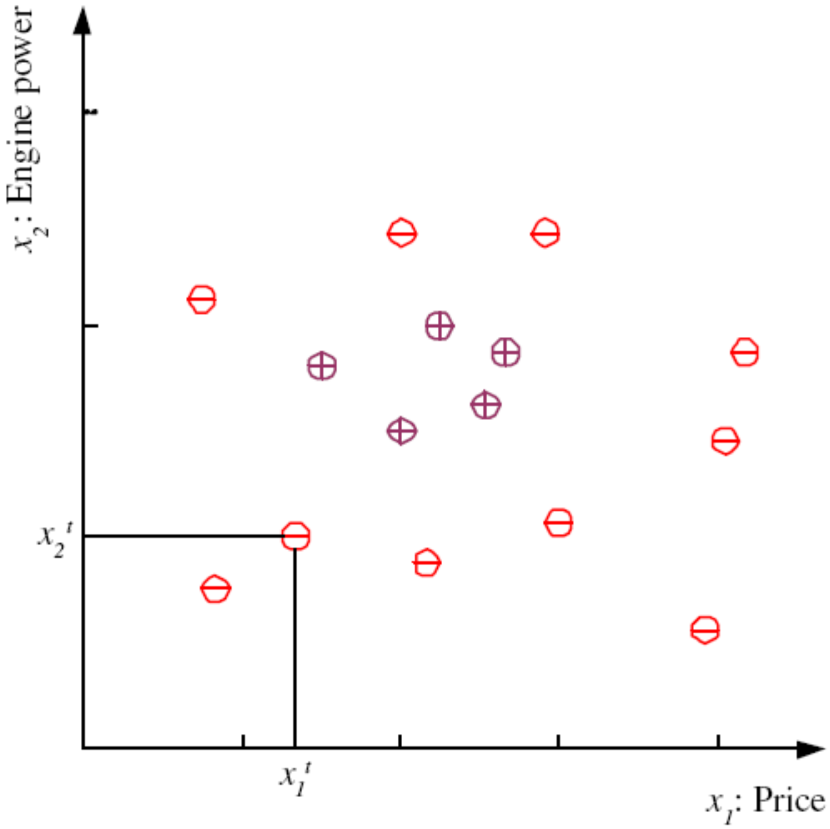
Example Supervised Learning: Classification

- Class C of a "family car"
  - Prediction: Is car x a family car?
  - Knowledge extraction: What do people expect from a family car?
- Output:

  Positive (+) and negative (–) examples
- Input representation by two features:

  x_1: price, x_2: engine power

Training set X



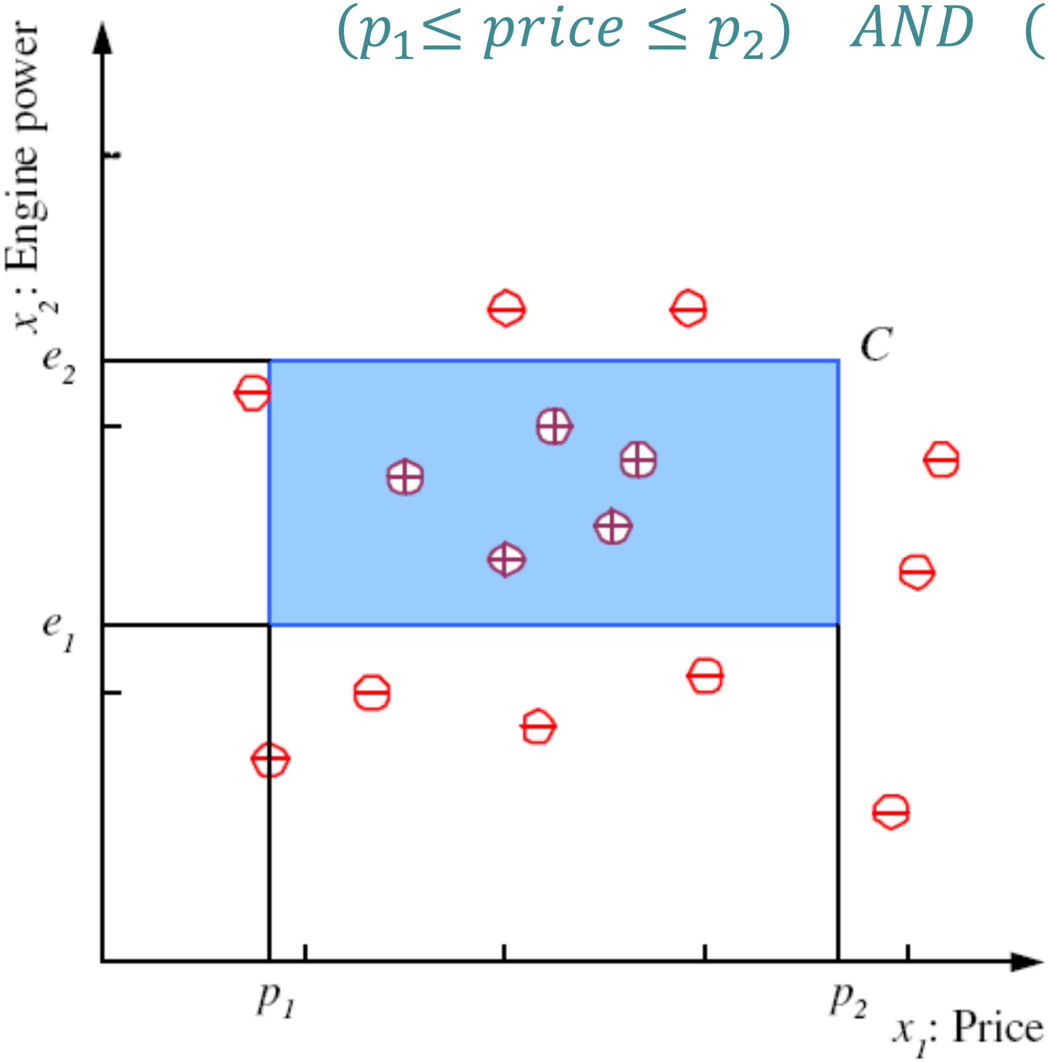
Labelled Data $X = \{ (x^t, r^t) \}_{t=1}^N$

Labels $r = \begin{cases} 1, & x \text{ is positive} \\ 0, & x \text{ is negative} \end{cases}$

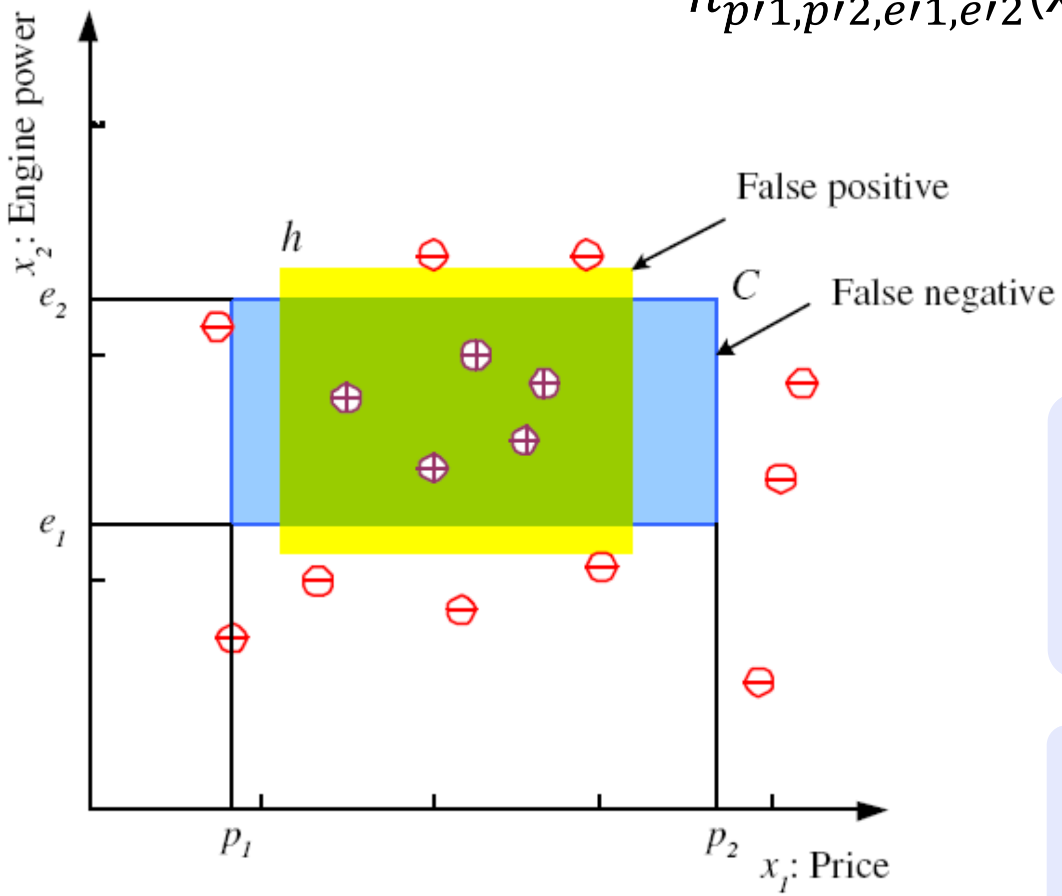
Feature vector $x = (x_1, x_2)$

Class C

$$(p_1 \leq price \leq p_2) \quad AND \quad (e_1 \leq engine\ power \leq e_2)$$



Hypothesis class H



$$h_{p'1,p'2,e'1,e'2}(\mathrm{x}) = \begin{cases} 1, & h \text{ classifies } x \text{ as positive} \\ 0, & h \text{ classifies } x \text{ as negative} \end{cases}$$

Error of h on X

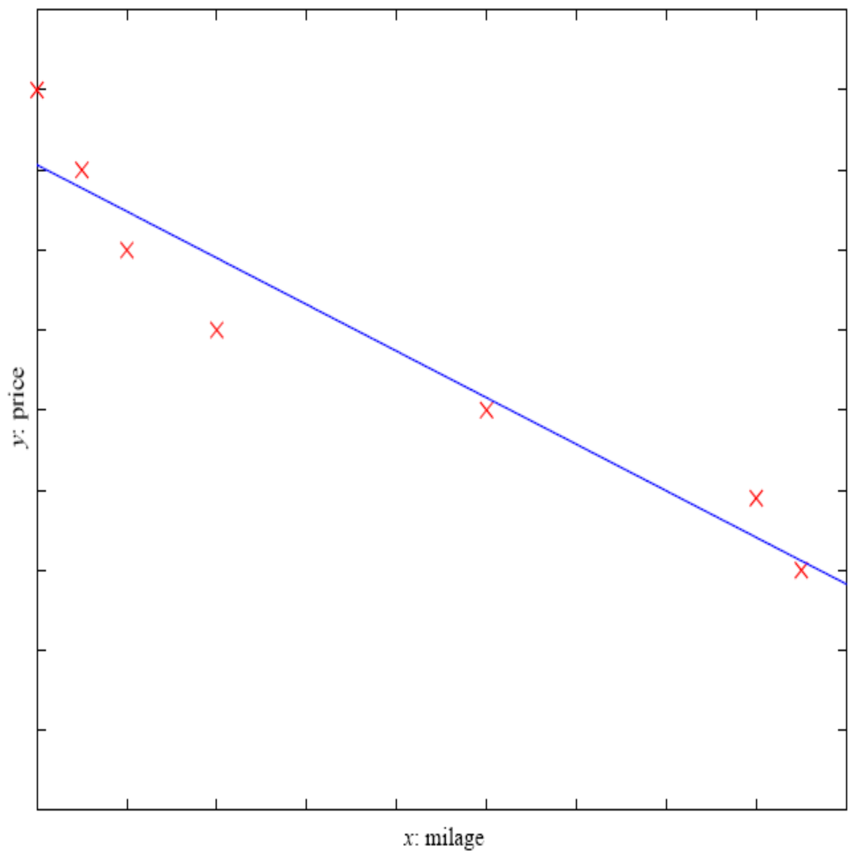
$$E(h|X) = (1/N) \sum_{t=1}^{N} (h(x^t) \neq y^t)$$

Optimization

$$argmin_{p_1' p_2' e_1' e_2'} E(h|X)$$

But how to find optimum?

Example Supervised Learning: Regression



Price of a used car

x : car attribute

y : price

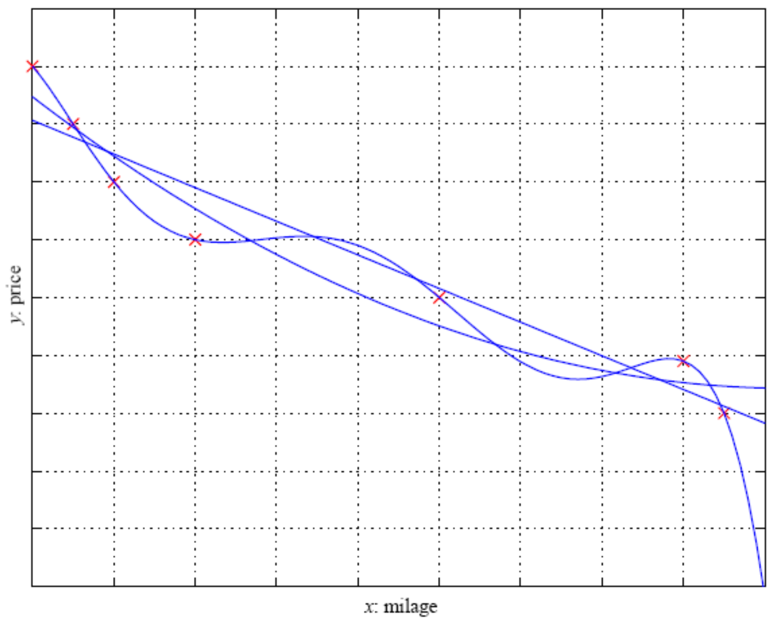
$\hat{y} = g(x \mid \theta)$: hypothesis

$g(\,)$: linear model

$$g(x) = w_1 x + w_0$$

θ: parameters (here w_1, w_2)

Example Supervised Learning: Regression



Price of a used car

x : car attribute

y : price

$\hat{y} = g\ (x \mid \theta\ )$: hypothesis

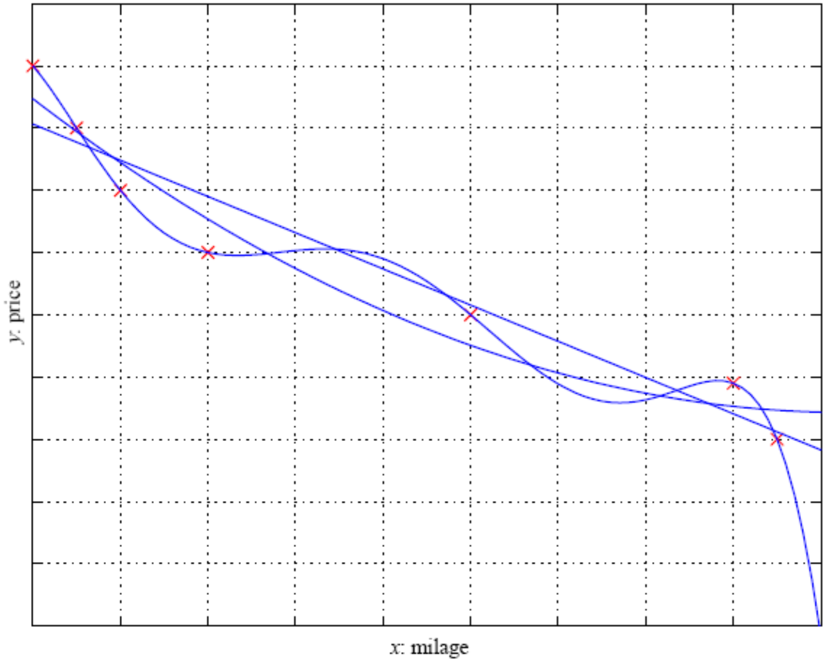
$g\ (\ )$: quadratic model

$$g(x) = w_2 x^2 + w_1 x + w_0$$

θ: parameters

(here w_0, w_1, w_2)

Example Supervised Learning: Regression



$$X = \{ (x^t, r^t) \}_{t=1}^{N} \quad r^t = f(x^t) \in \mathbb{R}$$

Mean squared error
for general and linear hypothesis g

$$E(g|X) = (1/N) \sum_{t=1}^{N} (g(x^t) - r^t)^2$$

$$E(w_1, w_0|X) = (1/N) \sum_{t=1}^{N} (w_1 x^t + w_0) - r^t)^2$$

Optimization:

$$\nabla E = \left(\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1} \right) = (0{,}0)$$

$$w_1 = \frac{\sum_t x^t r^t - \overline{x}\overline{r} N}{\sum_t (x^t)^2 - N\,\bar{x}^2}$$

$$w_0 = \bar{r} - w_1\,\bar{x}$$

Calculating the gradient ∇E analytically NOT feasible for thousands of parameters
- > Differentiable programming

DIFFERENTIABLE PROGRAMMING

What is Differentiable Programming (DP)?

„Yeah, Differentiable Programming is little more than a rebranding of the modern collection Deep Learning techniques, the same way Deep Learning was a rebranding of the modern incarnations of neural nets with more than two layers. The important point is that people are now building a new kind of software by assembling networks of parameterized functional blocks and by training them from examples using some form of gradient-based optimization….It's really very much like a regular program, except it's parameterized, automatically differentiated, and trainable/optimizable. ...

(Part of a post of Yann Lecun, somewhere in Facebook, found at https://gist.github.com/halhenke/872708ccea42ee8cafd950c6c2069814)

DP is a significant generalization of DL!

„Yeah, Differentiable Programming is little more than a rebranding of the modern collection Deep Learning techniques, the same way Deep Learning was a rebranding of the modern incarnations of neural nets with more than two layers. The important point is that people are now building a new kind of software by assembling networks of parameterized functional blocks and by training them from examples using some form of gradient-based optimization….It's really very much like a regular program, except it's parameterized, automatically differentiated, and trainable/optimizable. ...

(Part of a post of Yann Lecun, somewhere in Facebook, found at https://gist.github.com/halhenke/872708ccea42ee8cafd950c6c2069814)

What is Deep Learning (DL)?

V2

- Deep learning is based on function composition
  - Feedforward networks: $\mathbf{y} = f\,(g\,(\mathbf{x}\,,\,\theta_g\,),\,\theta_f\,)$
    Often with relatively simple functions
    (e.g. $f\,(\mathbf{x},\,\theta_f\,) = \sigma(\mathbf{x}^\top\theta_f\,)$)
  - Recurrent networks:
    $\mathbf{y}_t = f(\mathbf{y}_{t-1},\mathbf{x}_t,\theta) = f(f(\mathbf{y}_{t-2},\mathbf{x}_{t-1},\theta),\mathbf{x}_t,\theta) = \ldots$

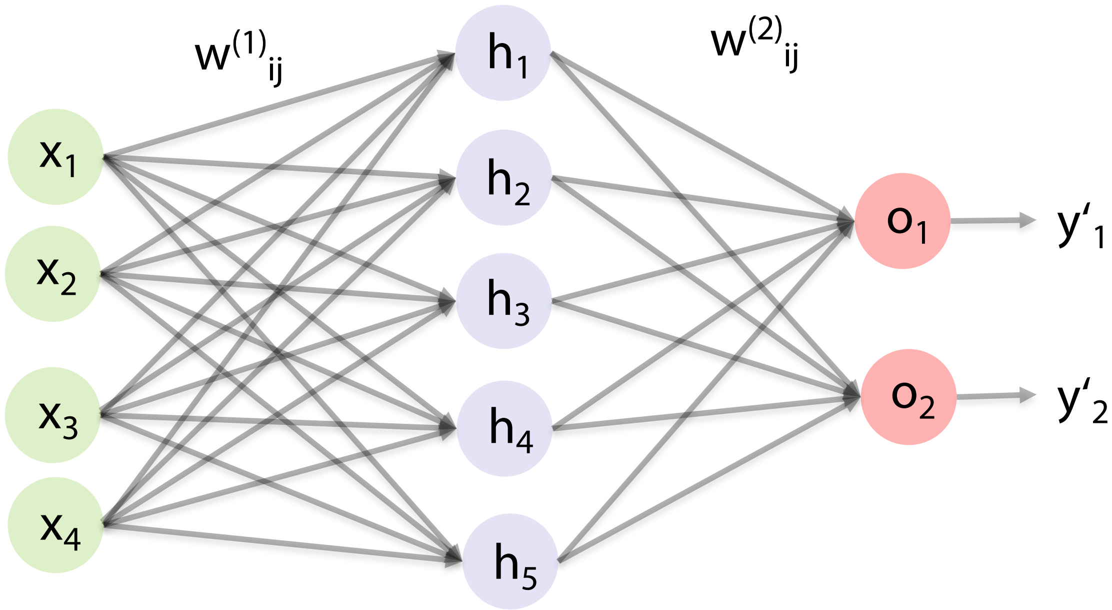
- In early days focus of DL on functions for classification

- Nowadays the functions are much more general in their inputs and outputs.

Network view of composed functions

V2

Input layer   Hidden layer(s)   Output layer



i: layer i
b_i: Bias in i
$W^{(i)}$: weight matrix in i
$\boldsymbol{\sigma}_i$: activation function in i

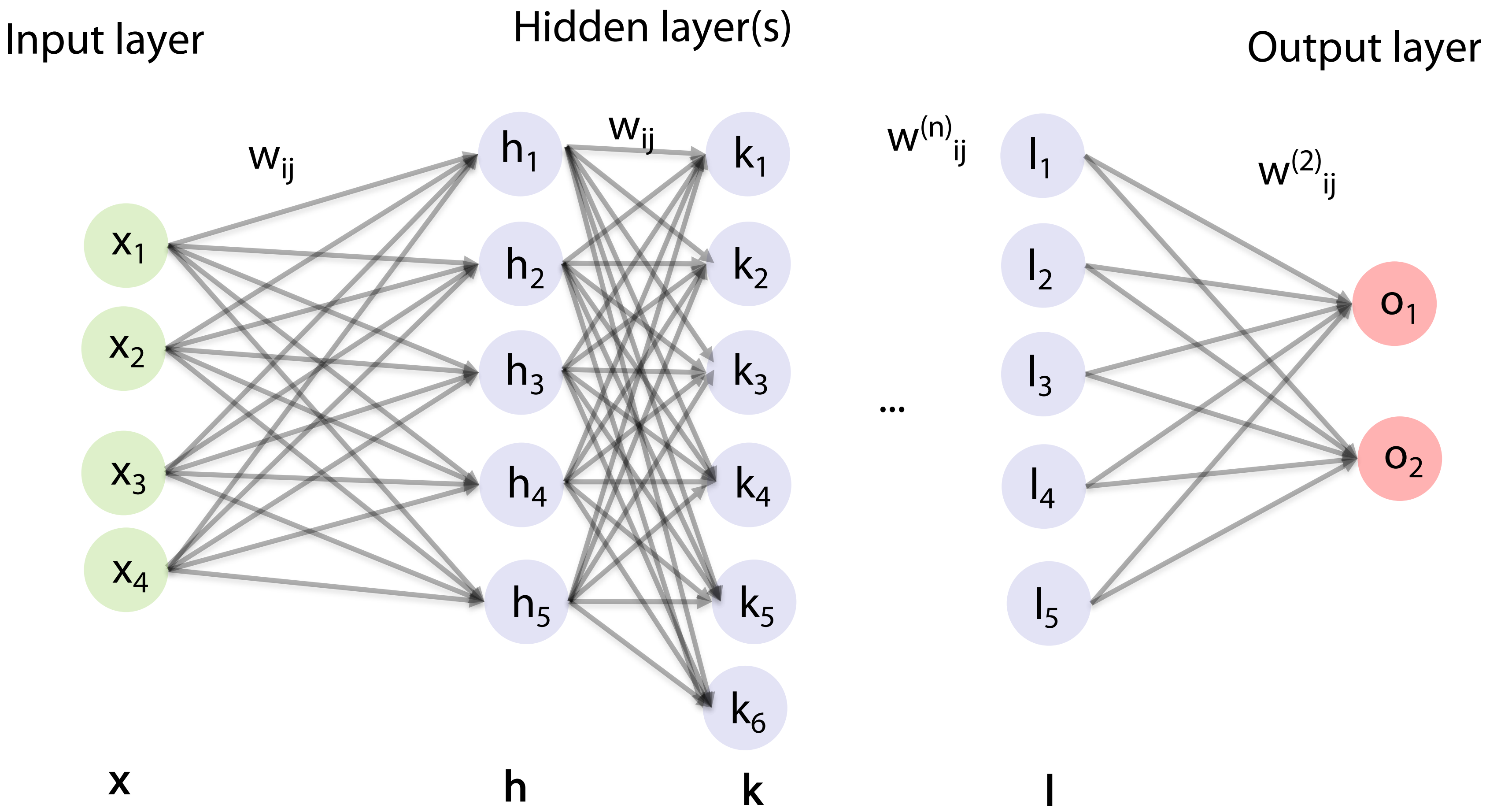
$\mathbf{x}$ $\mathbf{h}$ $\mathbf{y'}$ „States“

g   f   Functions

$$\mathbf{y'} = f\,(g\,(\mathbf{x}\,;\mathbf{W}^{(1)},\mathbf{b}_1);\mathbf{W}^{(2)},\mathbf{b}_2) = \boldsymbol{\sigma}_2\,(\mathbf{W}^{(2)}\,\boldsymbol{\sigma}_1(\mathbf{W}^{(1)}\,\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2)$$

Deep networks

V2



DP follows the gradient!

„Yeah, Differentiable Programming is little more than a rebranding of the modern collection Deep Learning techniques, the same way Deep Learning was a rebranding of the modern incarnations of neural nets with more than two layers. The important point is that people are now building a new kind of software by assembling networks of parameterized functional blocks and by training them from examples using some form of gradient-based optimization….It's really very much like a regular program, except it's parameterized, automatically differentiated, and trainable/optimizable. ...

(Part of a post of Yann Lecun, somewhere in Facebook, found at https://gist.github.com/halhenke/872708ccea42ee8cafd950c6c2069814)

Gradient Descent

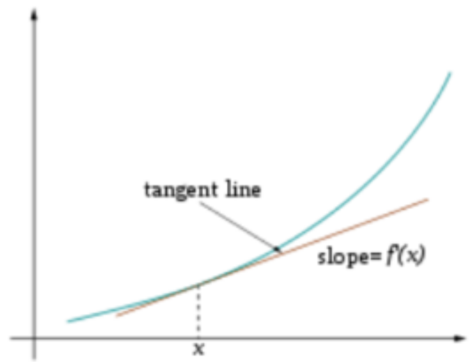
V2

- Total loss

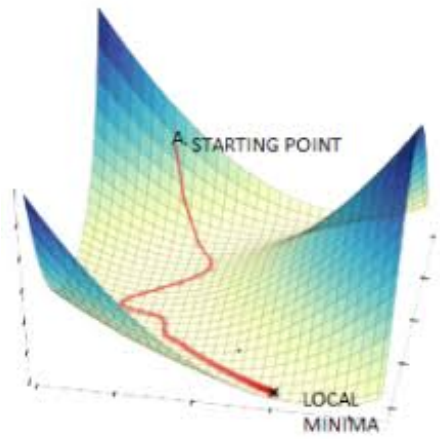
$$L = -\sum_{(x,y)\in D} l(g(x,\theta),y)$$

for some loss function l, dataset D

and model g with parameters θ



- Define how many passes (epochs) over the data to make
- learning rate η
- Gradient Descent: update θ by gradient in each epoch $\theta \leftarrow \theta - \eta \nabla_\theta L$



Backprop: efficient implementation of gradient descent

V2

Sample labeled data (**batch**) → **Forward** it through the network, get predictions → **Back-propagate** the errors → **Update** the network weights

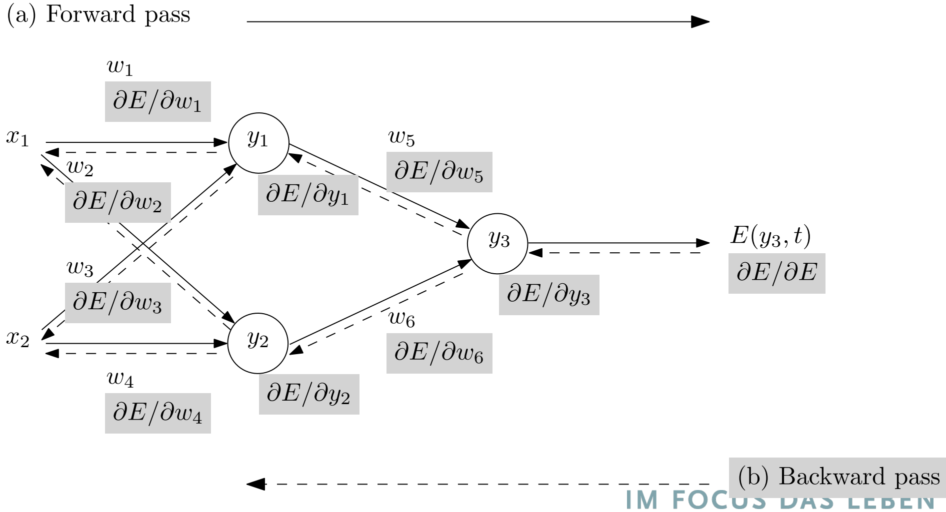
Backpropagation idea

- Generate **error signal** that measures difference between predictions and target values
- Use error signal to change the weights and get more accurate predictions backwards
- Underlying mathematics: chain rule

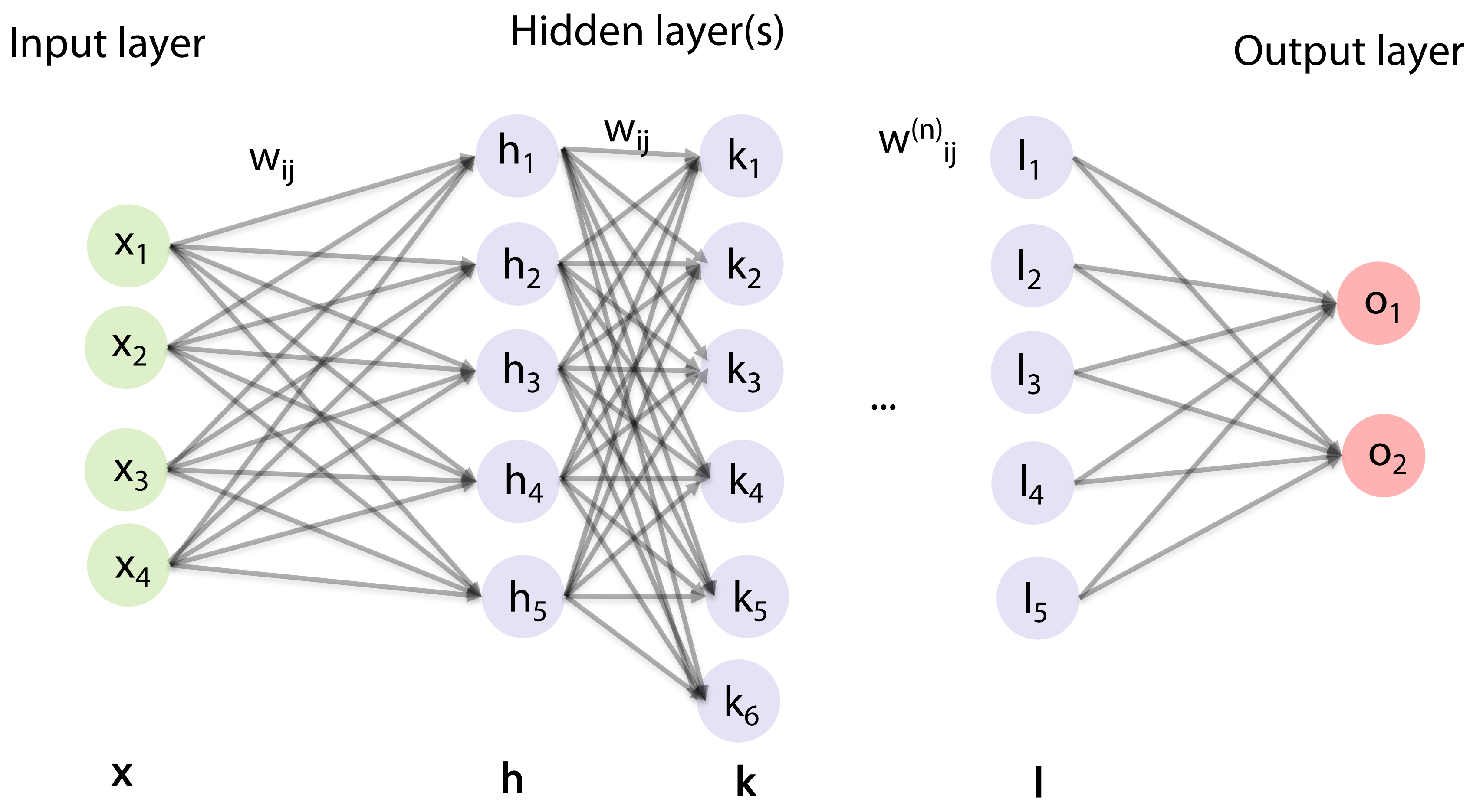
Chain rule (1-dim)

$$\frac{dh}{dx} = \frac{df}{dg}\frac{dg}{dx}$$

(for $h(x) = f(g(x))$)



Deep networks



Problem: Many, many parameters, no structure

What is Deep Learning?

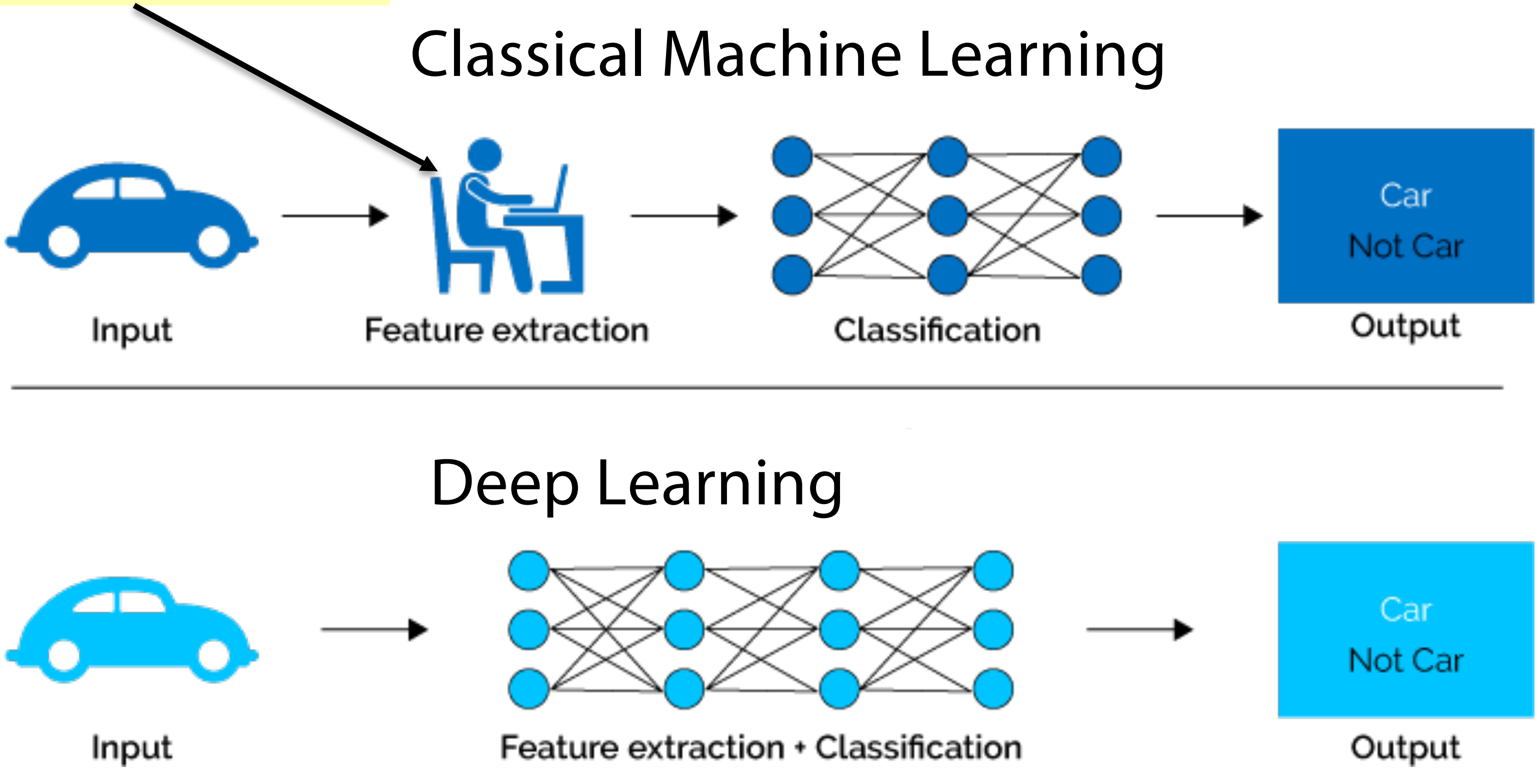
V3

- *Deep learning* systems are neural network models similar to those popular in the ’80s and ’90s, with:
    1. some architectural and algorithmic innovations (e.g. many layers, ReLUs, dropout, LSTMs)
    2. vastly larger data sets (web-scale)
    3. vastly larger-scale compute resources (GPU, cloud)
    4. much better software tools (Theano, Torch, TensorFlow)
    5. vastly increased industry investment and media hype

Deep Learning (ad 1.)

V3

Example family car:
we presumed features
price and mileage



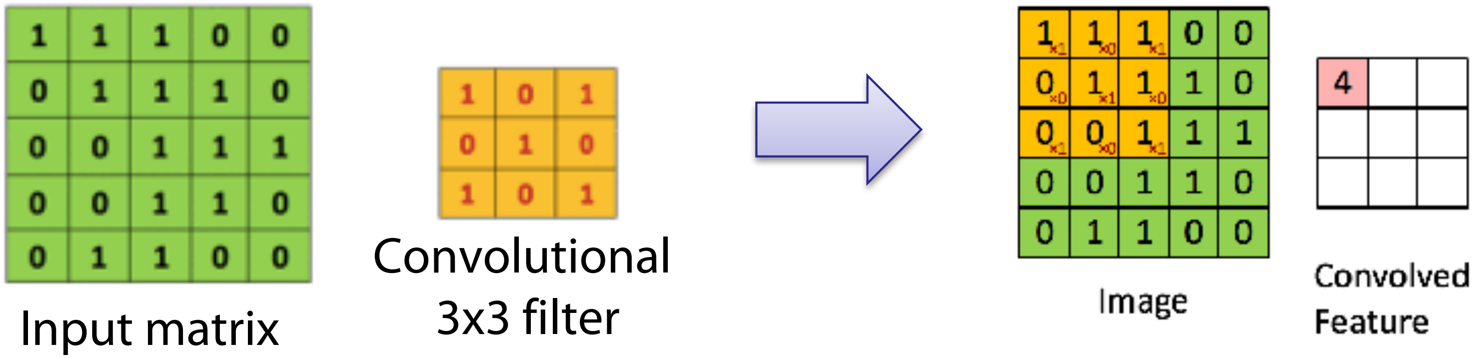
Adapted from https://www.xenonstack.com/blog/static/public/uploads/media/machine-learning-vs-deep-learning.png

Deep Learning (also ad 1.)

V3

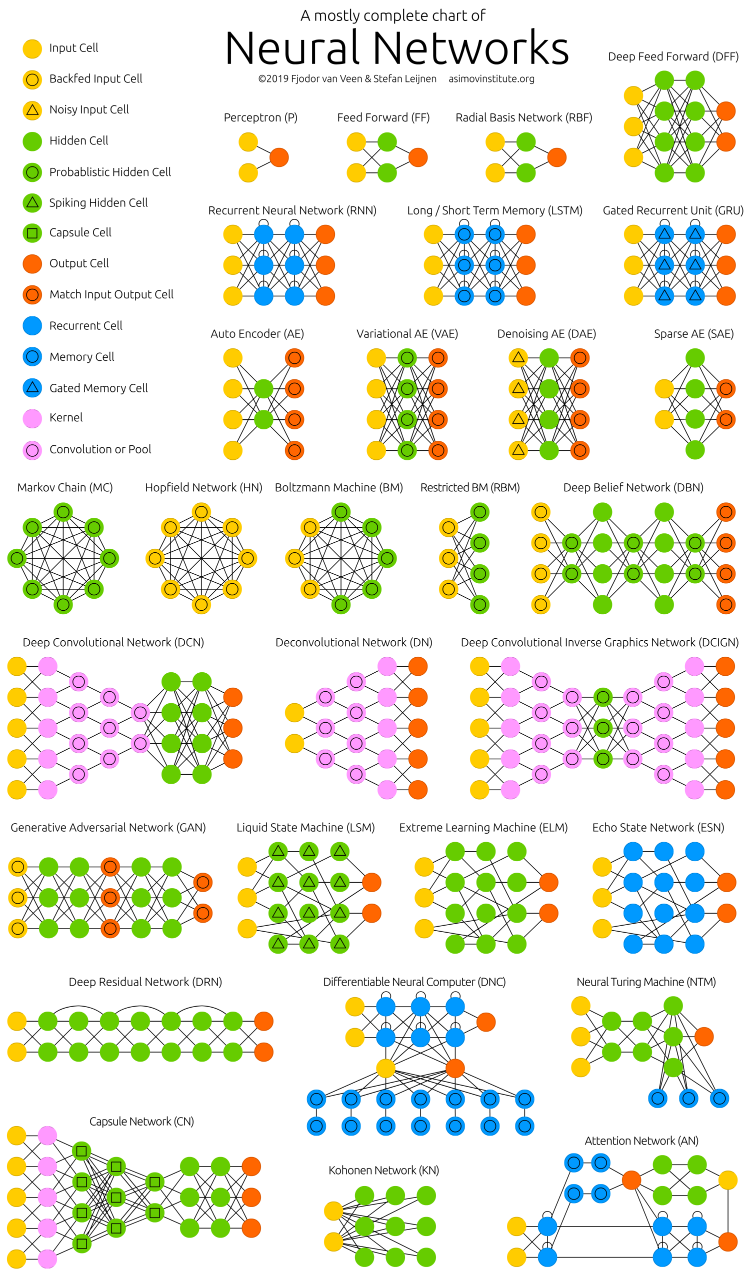
Example: Convolutional Neural Networks (CNN)

- More structure: local receptive fields
- Less parameters: weight tying, pooling



http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

V4



https://www.asimovinstitute.org/neural-network-zoo/

Why care about DL and study those structures?

Amazing performance on many benchmark tasks

DP uses automatic differentation (AD)

V5/6

„Yeah, Differentiable Programming is little more than a rebranding of the modern collection Deep Learning techniques, the same way Deep Learning was a rebranding of the modern incarnations of neural nets with more than two layers. The important point is that people are now building a new kind of software by assembling networks of parameterized functional blocks and by training them from examples using some form of gradient-based optimization….It's really very much like a regular program, except it's parameterized, automatically differentiated, and trainable/optimizable. ...

(Part of a post of Yann Lecun, somewhere in Facebook, found at https://gist.github.com/halhenke/872708ccea42ee8cafd950c6c2069814)

Automatic Differentiation (AD)

V5/6

- AD is a mix of
  - symbolic differentiation (SD) (rules s.a. chain rule, product rule)
  - numerical differentiation (ND): use $\frac{dy}{dx} \approx \frac{\Delta y}{\Delta x}$

$$\frac{d(f(x)\cdot g(x))}{dx} = \frac{d\,f(x)}{dx} g(x) + \frac{d\,g(x)}{dx} f(x) \quad \text{(Product rule)}$$

  - $h(x) := \ g(x) \cdot f(x)$
  - $\frac{dh(x)}{dx}$ and h have two components in common
  - This may also be the case für f.
  - Symbollicaly calculating f won‘t profit from common parts of f and $\frac{df(x)}{dx}$

V5/6

$l_1 = x$
$l_{n+1} = 4l_n(1-l_n)$

$f(x) = l_4 = 64x(1-x)(1-2x)^2(1-8x+8x^2)^2$

Manual Differentiation

$f'(x) = 128x(1-x)(-8+16x)(1-2x)^2(1-8x+8x^2)+64(1-x)(1-2x)^2(1-8x+8x^2)^2-64x(1-2x)^2(1-8x+8x^2)^2-256x(1-x)(1-2x)(1-8x+8x^2)^2$

Coding

```
f(x):
  v = x
  for i = 1 to 3
    v = 4*v*(1 - v)
  return v
```

or, in closed-form,

```
f(x):
 return 64*x*(1-x)*((1-2*x)^2)
   *(1-8*x+8*x*x)^2
```

Coding

Symbolic Differentiation of the Closed-form

```
f'(x):
 return 128*x*(1 - x)*(-8 + 16*x)
   *((1 - 2*x)^2)*(1 - 8*x + 8*x*x)
   + 64*(1 - x)*((1 - 2*x)^2)*((1
   - 8*x + 8*x*x)^2) - (64*x*(1 -
   2*x)^2)*(1 - 8*x + 8*x*x)^2 -
   256*x*(1 - x)*(1 - 2*x)*(1 - 8*x
   + 8*x*x)^2
```

f'(x_0) $= f'(x_0)$
Exact

Automatic Differentiation

Numerical Differentiation

```
f'(x):
  (v,dv) = (x,1)
  for i = 1 to 3
    (v,dv) = (4*v*(1-v), 4*dv-8*v*dv)
  return (v,dv)
```

f'(x_0) $= f'(x_0)$
Exact

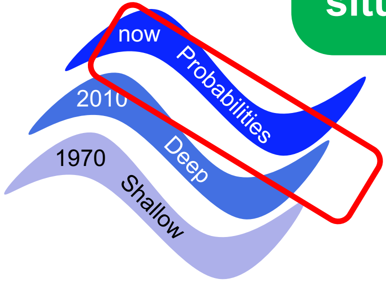
```
f'(x):
  h = 0.000001
  return (f(x + h) - f(x)) / h
```

f'(x_0) $\approx f'(x_0)$
Approximate

PROBABILITIES

„ **The third wave of differentiable programming**

Getting deep systems that know when they do not know and, hence, recognise new situations and adapt to them



" 1)

Kristian Kersting - Sum-Product Networks: The Third Wave of Differentiable Programming

---

1) Yes, a slide, quoting a slide

Problems with deep (neural) networks (Ghahramani)

- Very data hungry (e.g. often millions of examples)
- Very compute-intensive to train and deploy (cloud GPU resources)
- Poor at representing uncertainty
- Easily fooled by adversarial examples
- Finicky to optimise: non-convex + choice of architecture, learning procedure, initialisation, etc, require expert knowledge and experimentation
- Uninterpretable black-boxes, lacking in trasparency, difficult to trust

Bayes rule to rule it all ...

V7

- If we use the mathematics of probability theory to express all forms of uncertainty and noise associated with our model...
- ...then inverse probability (i.e. Bayes rule) allows us to infer unknown quantities, adapt our models, make predictions and learn from data.

$$P(H|D) = \frac{P(D|H) \cdot P(H)}{P(D)} = \frac{P(D|H) \cdot P(H)}{\sum_h P(D|h)P(h)}$$

H = hypothesis, model

D = data, observation

Bayes Rule

Probabilistic graphical models

V7

Encode efficiently full joint probabilities

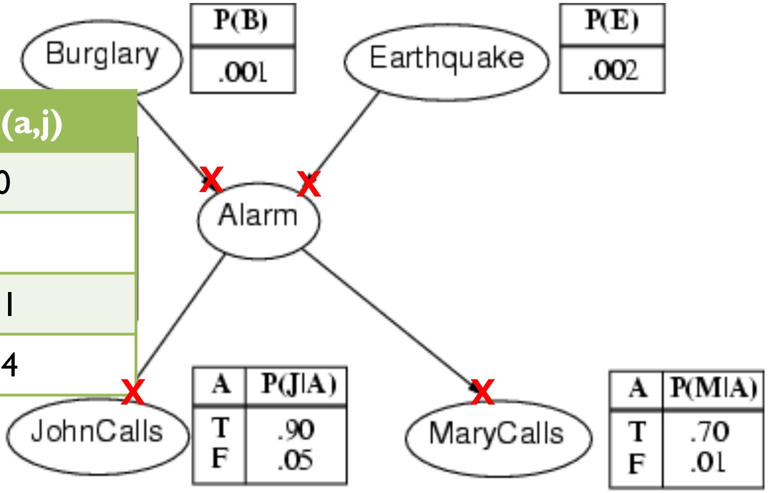
- Directed graphs

  (Bayesian networks,

  Hidden Markov models ...)
- undirected graphs

  (Markov networks...)
- Mixed models
- Factor graphs

A	J	φ(a,j)
F	F	20
F	T	1
T	F	0.1
T	T	0.4



Requires Normalization

$$P(B=b, E=e, A=a, j, m) =$$

$$\frac{1}{Z}\phi_{JA}(a,j)\phi_{MA}(a,m)\phi_{AB}(a,b), \phi_{AE}(a,e)\phi_{B}(b)$$

$$Z = \sum_x \prod_j \phi_j$$

Partition function

PROBABILISTIC PROGRAMMING

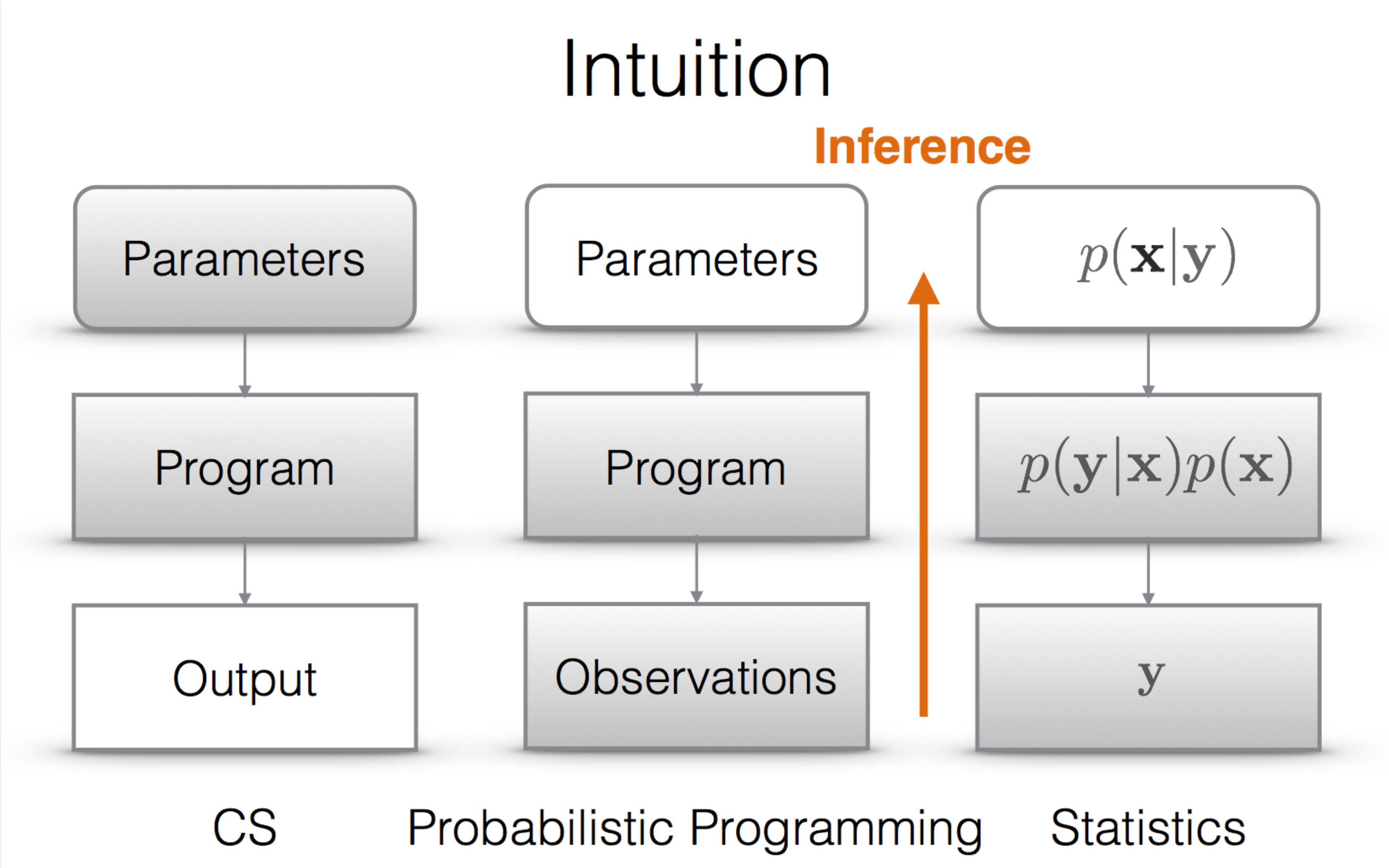
Why then not stick to probabilities

V8

- Problem 1: Probabilistic model development and the derivation of inference algorithms is time-consuming and error-prone.
- Problem 2: Exact (and approximate inference) hard due to normalization: partition function Z)
- Solution to 1
  - Develop Probabilistic Programming Languages for expressing probabilistic models as computer programs that generate data (i.e. simulators).
  - Derive Universal Inference Engines for these languages that do inference over program traces given observed data (Bayes rule on computer programs).

Comparison

V8



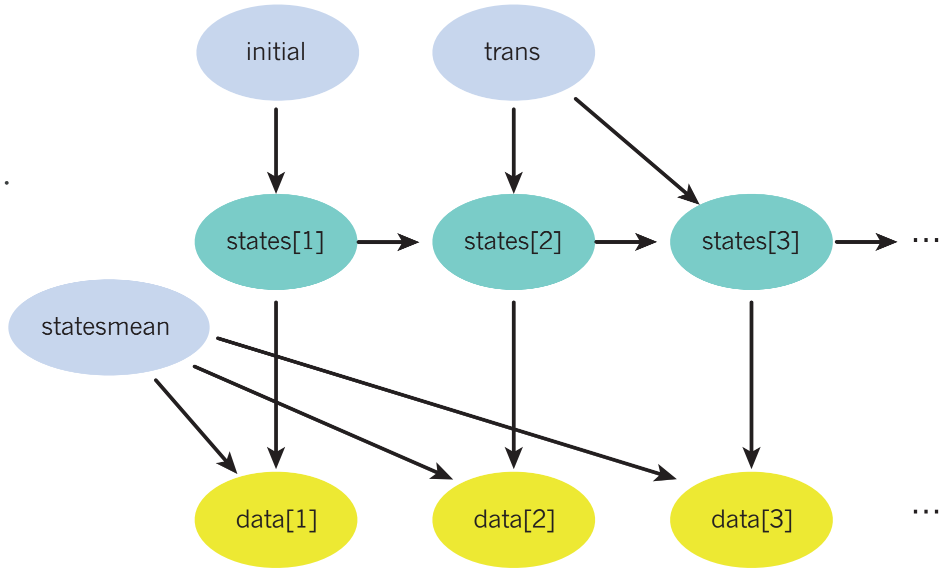
Ex: F. Wood: Probabilistic Programming, PPAML Summer School, Portland 2016

Probabilistic Programming Example

V8

```
statesmean = [-1, 1, 0]  # Emission parameters.
initial    = Categorical([1.0/3, 1.0/3, 1.0/3]) # Prob distr of state[1].
trans      = [Categorical([0.1, 0.5, 0.4]), Categorical([0.2, 0.2, 0.6]),
               Categorical([0.15, 0.15, 0.7])]   # Trans distr for each state.
data       = [Nil, 0.9, 0.8, 0.7, 0, -0.025, -5, -2, -0.1, 0, 0.13]

@model hmm begin # Define a model hmm.
 states = Array(Int, length(data))
 @assume(states[1] ~ initial)
 for i = 2:length(data)
   @assume(states[i] ~ trans[states[i-1]])
   @observe(data[i]  ~ Normal(statesmean[states[i]], 0.4))
 end
 @predict states
end
```



Hidden markov model in Julia

ADEQUATE DEEP STRUCTURES

Problem 2 of probabilistic graphical models

- Exact (and even approxiate) inference not tractable for general probabilistic models (problem: normalization function Z).

- Restricting the models in expressivity is possible (thin junction trees and so on) - but not desirable

- Find a better compromise of expressivity and feasibility: sum-product networks/probabilisitc boolean circuits

Why Is Inference Hard?

$$P(X_1, \dots Xn) = \frac{1}{Z} \prod_j \phi_j (X_1, \dots Xn)$$

- Bottleneck: Summing out variables
- E.g.: Partition function

  Sum of exponentially many products

$$Z = \sum_x \prod_j \phi_j$$

Alternative Representation

X_1	X_2	$P(X)$
1	1	0.4
1	0	0.2
0	1	0.1
0	0	0.3

$$\begin{aligned} P(X) = {} & 0.4 \cdot X_1 \cdot X_2 \\ & + 0.2 \cdot X_1 \cdot \overline{X}_2 \\ & + 0.1 \cdot \overline{X}_1 \cdot X_2 \\ & + 0.3 \cdot \overline{X}_1 \cdot \overline{X}_2 \end{aligned}$$

Network Polynomial [Darwiche, 2003]

Sum Out Variables

X_1	X_2	$P(X)$
1	**1**	**0.4**
1	**0**	**0.2**
0	1	0.1
0	0	0.3

$$\boxed{e: X_1 = 1}$$

$$\begin{aligned} P(e) = & \mathbf{0.4 \cdot X_1 \cdot X_2} \\ & \mathbf{+ 0.2 \cdot X_1 \cdot \overline{X}_2} \\ & + 0.1 \cdot \overline{X}_1 \cdot X_2 \\ & + 0.3 \cdot \overline{X}_1 \cdot \overline{X}_2 \end{aligned}$$

Set $X_1 = 1, \overline{X}_1 = 0, \boxed{X_2 = 1, \overline{X}_2 = 1}$

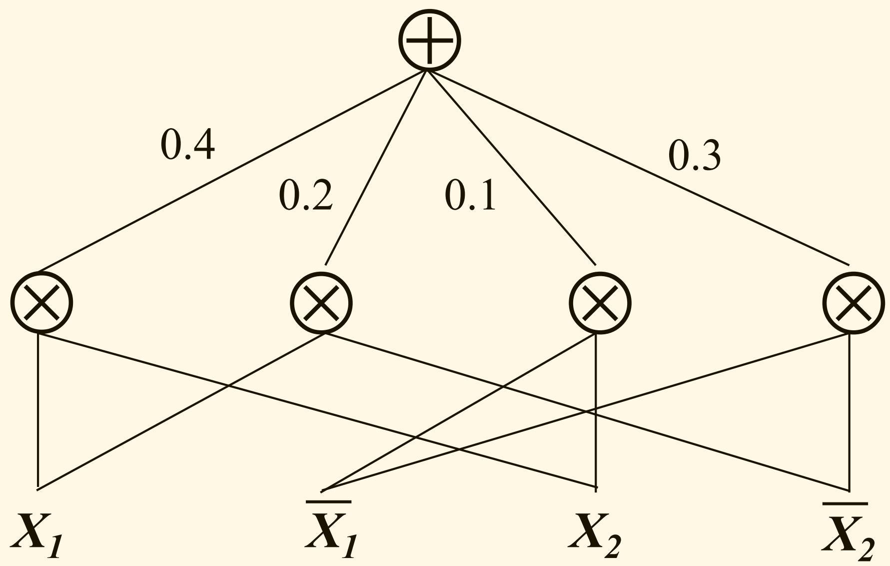
Easy: Set both indicators to 1

Easy: Partition function: Set all indicators to 1

Graphical Representation

V9/10

X_1	X_2	$P(X)$
1	1	0.4
1	0	0.2
0	1	0.1
0	0	0.3

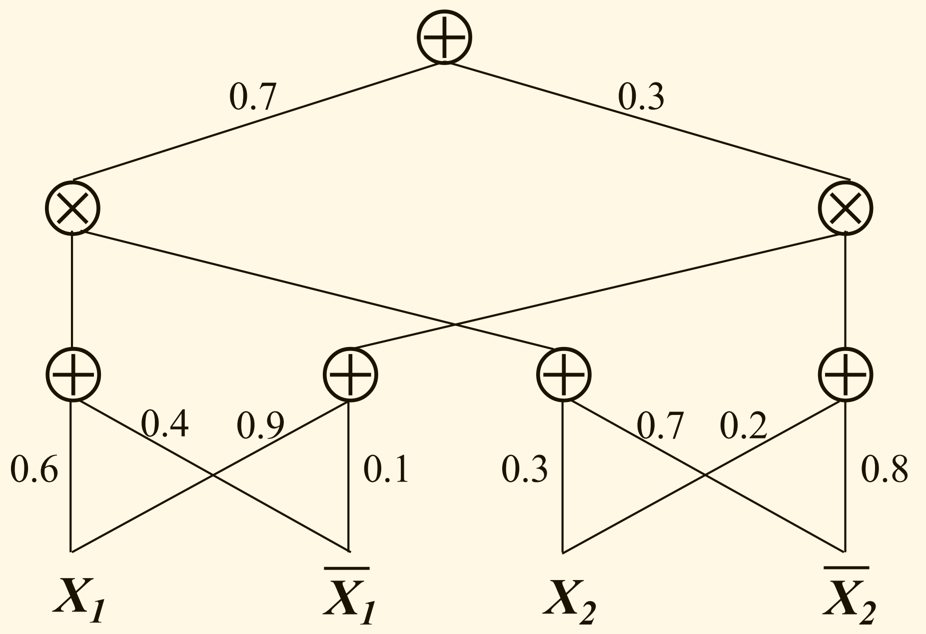


But in general may lead to exponentially large networks (e.g. parity).
Solution: Make a deep dive (reuse computations)
with Sum-Product networks

Sum-Product Networks (SPNs)

- Rooted DAG
- Nodes: Sum, product, input indicator
- Weights on edges from sum to children

- More general class: Probabilistic Boolean Circuits



NEARLY THE END

Topic progress of course in short

- It‘ from discriminative to generative models
- It‘s from pure functions to algorithms to algorithms over semi-declarative structures (and some logic)
- It‘s from non-probabilities to probabilities (and some logic)

Uhhh, a lecture with a hoepfully useful

APPENDIX

Todays lecture is based on the following slides

- Jonathon Hare: Lecture 1,2 of course „COMP6248 Differentiable Programming (and some Deep Learning") http://comp6248.ecs.soton.ac.uk/
- Zoubin Ghahramani: Probabilistic Machine Learning and AI, Microsoft AI Summer School Cambridge 2017 http://mlss.tuebingen.mpg.de/2017/speaker_slides/Zoubin1.pdf
- Hoifung Poon: Sum-Product Networks: A New Deep Architecture https://www.microsoft.com/en-us/research/wp-content/uploads/2017/05/spn11.pdf
- E. Alpaydin: Course on machine learning, introductory slides, https://www.cmpe.boun.edu.tr/~ethem/i2ml2e/2e_v1-0/i2ml2e-chap1-v1-0.pptx
- I. Lorentzou: Introduction to Deep Learnin, link
- F. Wood: Probabilistic Programming, PPAML Summer School, Portland 2016, link

Color Convention in this course

- Formulae, when occurring inline
- Newly introduced terminology and definitions
- Important results (observations, theorems) as well as emphasizing some aspects
- Examples are given with standard orange with possibly light orange frame
- Comments and notes
- Algorithms

Books for topics covered in this lecture (1)

- Nielsen: Neural Networks and Deep Learning. http://neuralnetworksanddeeplearning.com/
- Zhang et al.: Dive into Deep Learning https://d2l.ai/
- I. Goodfellow, Y. Bengio, and A. Courville. Deep Learning. MIT Press, 2016.
- D. Koller and N. Friedman. Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning. The MIT Press, 2009.
- L. D. Raedt, K. Kersting, and S. Natarajan. Statistical Relational Artificial Intelligence: Logic, Probability, and Computation. Morgan & Claypool Publishers, 2016.

Books for topics covered in this lecture (2)

- J.-W. van de Meent, B. Paige, H. Yang, and F. Wood. An Introduction to Probabilistic Programming. arXiv e-prints, arXiv:1809.10756, Sept. 2018.
- U. Naumann. The Art of Differentiating Computer Programms. Siam, 2012.
- K. Murphy. Machine Learning: A Probabilistic Perspective. Adaptive Computation and Machine Learning series. MIT Press, 2012.
- S. J. Russell and P. Norvig. Artificial Intelligence – A Modern Approach. Prentice Hall, 1995.