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) 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 optimization 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

1. Machine Learning
2. Differentiable Programming and
3. Probabilistic Programming for

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 optimization 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
Machine learning (ML) is programming algorithms for
- optimizing a performance criterion
- using example (training) data
- by constructing generalizable 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).

https://simple.wikipedia.org/wiki/Reinforcement_learning
Subtypes of unsupervised I. (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 $X$

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

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

\((p_1 \leq \text{price} \leq p_2) \quad \text{AND} \quad (e_1 \leq \text{engine power} \leq e_2)\)
Hypothesis class $H$

$$h_{p_1, p_2, e_1, e_2}(x) = \begin{cases} 1, & h \text{ classifies } x \text{ as positive} \\ 0, & h \text{ classifies } x \text{ as negative} \end{cases}$$

Error of $h$ on $X$

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

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

\[ y = g(x | \theta): \] hypothesis

\[ g(x) = w_1 x + w_0 \] linear model

\[ \theta: \] parameters (here \( w_1, w_2 \))
Example Supervised Learning: Regression

Price of a used car

\[ x : \text{car attribute} \]
\[ y : \text{price} \]
\[ \hat{y} = g ( x | \theta ) : \text{hypothesis} \]
\[ g ( ) : \text{quadratic model} \]

\[ g(x) = w_2 x^2 + w_1 x + w_0 \]

\[ \theta : \text{parameters} \]

(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 \( g \)

\[
E(g|X) = \frac{1}{N} \sum_{t=1}^{N} (g(x^t) - r^t)^2
\]

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

Optimization:

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

\[
w_1 = \frac{\sum_t x^t r^t - \bar{x} \bar{r} N}{\sum_t (x^t)^2 - N \bar{x}^2}
\]

\[
w_0 = \bar{r} - w_1 \bar{x}
\]

Calculating the gradient \( \nabla E \) analytically NOT feasible for thousands of parameters

- \( \rightarrow \) 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)?

- Deep learning is based on function composition
  - Feedforward networks: \( y = f(g(x, \theta_g), \theta_f) \)
    Often with relatively simple functions
    (e.g. \( f(x, \theta_f) = \sigma(x^T \theta_f) \))
  - Recurrent networks:
    \( y_t = f(y_{t-1}, x_t, \theta) = f(f(y_{t-2}, x_{t-1}, \theta), x_t, \theta) = \ldots \)

- 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

\[ y' = f \left( g \left( x; W^{(1)}, b_1 \right); W^{(2)}, b_2 \right) = \sigma_2 \left( \sigma_1 \left( W^{(1)} x + b_1 \right) + b_2 \right) \]
Deep networks

Input layer

Hidden layer(s)

Output layer

\[ w_{ij} \]

\[ h_1 \]
\[ h_2 \]
\[ h_3 \]
\[ h_4 \]
\[ h_5 \]

\[ k_1 \]
\[ k_2 \]
\[ k_3 \]
\[ k_4 \]
\[ k_5 \]

\[ l_1 \]
\[ l_2 \]
\[ l_3 \]
\[ l_4 \]
\[ l_5 \]

\[ o_1 \]
\[ o_2 \]
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

- **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 \( \theta \)
- Define how many passes (epochs) over the data to make
- learning rate \( \eta \)
- **Gradient Descent:** update \( \theta \) by gradient in each epoch \( \theta \leftarrow \theta - \eta \nabla_{\theta} L \)
Backprop: efficient implementation of gradient descent

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)

\[
\frac{dh}{dx} = \frac{df}{dg} \cdot \frac{dg}{dx}
\]

(for \( h(x) = f(g(x)) \))
Deep networks

Problem: Many, many parameters, no structure
What is Deep Learning?

• 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.)

Example family car: we presumed features price and mileage

Classical Machine Learning

Input → Feature extraction → Classification → Output

Deep Learning

Input → Feature extraction + Classification → Output

Adapted from https://www.xenonstack.com/blog/static/public/uploads/media/machine-learning-vs-deep-learning.png
Deep Learning (also ad 1.)

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
Why care about DL and study those structures?

Amazing performance on many benchmark tasks

https://www.asimovinstitute.org/neural-network-zoo/
“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/872708ccea42ee8caf9d50c6c2069814)
Automatic Differentiation (AD)

- AD is a mix of
  - symbolic differentiation (SD) (rules s.a. chain rule, product rule)
  - numerical differentiation (ND): use \( \frac{dy}{dx} \approx \frac{\Delta y}{\Delta x} \)
  \[
  \frac{d(f(x) \cdot g(x))}{dx} = \frac{df(x)}{dx} g(x) + \frac{dg(x)}{dx} f(x) \quad \text{(Product rule)}
  \]
  - \( h(x) := g(x) \cdot f(x) \)
  - \( \frac{dh(x)}{dx} \) and \( h \) have two components in common
  - This may also be the case für \( f \).
  - Symbolically calculating \( f \) won’t profit from common parts of \( f \) and \( \frac{df(x)}{dx} \)
The range of approaches for differentiating mathematical expressions and computer code, looking at the example of a truncated logistic map (upper left). Symbolic differentiation (center right) gives exact results but requires closed-form input and suffers from expression swell; numerical differentiation (lower right) has problems of accuracy due to round-off and truncation errors; automatic differentiation (lower left) is as accurate as symbolic differentiation with only a constant factor of overhead and support for control flow.
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) 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 transparency, difficult to trust
Bayes rule to rule it all ...

- 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

Encode efficiently **full joint probabilities**

- Directed graphs
  (Bayesian networks, Hidden Markov models ...)
- Undirected graphs
  (Markov networks...)
- Mixed models
- Factor graphs

![Diagram of a Bayesian network](image)

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

- 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

Probabilistic Programming Example

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 approximate) 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/probabilisitic boolean circuits
Why Is Inference Hard?

\[ P(X_1, \ldots, X_n) = \frac{1}{Z} \prod_j \phi_j(X_1, \ldots, X_n) \]

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

Sum of exponentially many products

\[ Z = \sum_x \prod_j \phi_j \]
Alternative Representation

\[
P(X) = 0.4 \cdot X_1 \cdot X_2 \\
+ 0.2 \cdot X_1 \cdot \overline{X}_2 \\
+ 0.1 \cdot \overline{X}_1 \cdot X_2 \\
+ 0.3 \cdot \overline{X}_1 \cdot \overline{X}_2
\]

<table>
<thead>
<tr>
<th>(X_1)</th>
<th>(X_2)</th>
<th>(P(X))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0.1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0.3</td>
</tr>
</tbody>
</table>

Network Polynomial [Darwiche, 2003]
Sum Out Variables

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$P(X)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0.1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0.3</td>
</tr>
</tbody>
</table>

$e$: $X_1 = 1$

\[
P(e) = 0.4 \cdot X_1 \cdot X_2 + 0.2 \cdot X_1 \cdot \overline{X}_2 + 0.1 \cdot \overline{X}_1 \cdot X_2 + 0.3 \cdot \overline{X}_1 \cdot \overline{X}_2
\]

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

Easy: Set both indicators to 1

Easy: Partition function: Set all indicators to 1
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’s 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)
Uhuh, a lecture with hopefully useful

APPENDIX
Today's 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  
- Hoifung Poon: Sum-Product Networks: A New Deep Architecture  
- 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 Learning, 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)

- Zhang et al.: Dive into Deep Learning https://d2l.ai/
Books for topics covered in this lecture (2)