Machine Intelligence:: Deep Learning
Week 6

Beate Sick, Loran Avci, Oliver Dürr

Part II

Winterthur, 30. March. 2021

Count Distributions in the real world

Building state of the art networks

Two Design choices

1. Choose the right network architecture to exploit structure of your problem ("inductive bias")
   - Images CNN
   - Sequences RNN or 1D convolution

2. Choose the right CPD for your data

- Train with NLL
- Evaluate performance on validation with NLL

Two state-of-the art models PixelCNN and Wavenet have been improved by using *discretized logistic mixture* distributions as CPD.

PixelCNN and Wavenet In/Output

- Both are autoregressive models $p(x_t | x_{t-1}, \dots x_0)$
  - Wavenet
    - Predicts the next audio sample x_t from all previous $x_{t-1}, \dots x_0$
    - x_t can take a number between 0 and 2^16-1 = 65,535
  - PixelCNN
    - Predicts the next pixel x_t from all previous $x_{t-1}, \dots x_0$
    - x_t can take a value between 0 and 2^8-1 = 255

Wavenet architecture (see lecture 06_sequences)

PixelCNN and Wavenet In/Output

Modelling $p(x_t | \, x_{t-1}, \dots x_0)$

- Naive approach output probabilities for the different classes
  - This has been done in first versions
  - Does not take nature of the output data into account

- Better find a *flexible* distribution to model the discrete data between 0 and say 255
  - Both architectures have been improved by mixtures of discrete logistics

We now have a look at the mixtures of discrete logistics.

The code for the figures is in:
https://github.com/tensorchiefs/dl_book/blob/master/chapter_06/nb_ch06_01.ipynb

Making sense of discretized logistic mixture

- The base distribution: the logistic with 2 parameters
  - loc kind of center
  - scale spread

Figure 6.3 Three logistic functions created using tfd.Logistic(loc=1, scale=scale) with values of 0.5, 1.0, and 2.0 for the scale parameter. On the left is the probability density function (PDF) and on the right, the cumulative probability density function (CDF).

- The CFD has the same functional form as the sigmoid activation
- Still not discrete

Making sense of discretized logistic mixture

- The quantized (discretized) base distribution the logistic

Making sense of discretized logistic mixture

- Mixing several (here 2) discretized logistics

Figure 6.5 The resulting discrete distribution when mixing two logistic distributions (see listing 6.2 for the code that produces these plots)

Number of components, which need to be determined by network?

- 2 (or 1) for mixing
- 2 for scale
- 2 for location

Case Study for discretized logistic mixture

The original PixelCNN++ and Wavenet take too long to train. The fish data is too small for deep learning models. Let's use a large count data set.

Goal: Probabilistic model for deer activity conditioned on the time (in day and year).

wild	year	time	daytime	weekday
0	2002.0	0.000000	night.am	Sunday
0	2002.0	0.020833	night.am	Sunday
...	....	....	....	...
1	2002.0	0.208333	night.am	Sunday
0	2002.0	0.229167	pre.sunrise.am	Sunday
0	2002.0	0.270833	pre.sunrise.am	Sunday

The columns have the following meaning:

wild: the number deers killed in a road accident in Bavaria

year: the year (from 2002 to 2009 in the training set from 2010 to 2011 for the test set)

time: the number of days to the first event. These numbers are measured in fractions of a day.

Data on deer related car accidents in the years 2002 until 2011 in Bavaria, Germany.
Target variable (wild): use number of deers killed during 30 minute period as surrogate

Modeling count data

Goal:
Predict CPD for y=#deers-killed-in-30min, given x (time and derived variables).
Possible CPD models:

$$(y \mid x) \sim \mathrm{Pois}(\lambda_x)$$

$$(y \mid x) \sim \mathrm{ZIP}({}^{z}p_x, \lambda_x)$$

$$(y \mid x) \sim \mathrm{discretizedLogisticMix}({}^{1}p_x, {}^{2}p_x\, {}^{3}p_x, {}^{1}\mu_x, {}^{2}\mu_x, {}^{3}\mu_x, {}^{1}\sigma_x, {}^{2}\sigma_x, {}^{3}\sigma_x)$$

Marginal distribution of deer killed in 30min

Code

```
def quant_mixture_logistic(out, bits=8, num=3):
    loc, un_scale, logits = tf.split(out,#A
                                     num_or_size_splits=num,
                                     axis=-1)
    scale = tf.nn.softplus(un_scale) #B
    discretized_logistic_dist = tfd.QuantizedDistribution(
      distribution=tfd.TransformedDistribution( #C
          distribution=tfd.Logistic(loc=loc, scale=scale),
          bijector=tfb.AffineScalar(shift=-0.5)),
      low=0.,
      high=2**bits - 1.
    )
    mixture_dist = tfd.MixtureSameFamily(#D
        mixture_distribution=tfd.Categorical(logits=logits),
        components_distribution=discretized_logistic_dist)
    return mixture_dist

```

```
inputs = tf.keras.layers.Input(shape=(100,))
h1 = Dense(10, activation='tanh')(inputs)
out = Dense(6)(h1) #E
p_y = tfp.layers.DistributionLambda(quant_mixture_logistic)(out)
```

https://github.com/tensorchiefs/dl_book/blob/master/chapter_06/nb_ch06_01.ipynb

Architectures

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, 16)]              0
_________________________________________________________________
dense_10 (Dense)             (None, 100)               1700
_________________________________________________________________
dense_11 (Dense)             (None, 100)               10100
_________________________________________________________________
dense_12 (Dense)             (None, 10)                1010
_________________________________________________________________
dense_13 (Dense)             (None, 9)                 99
_________________________________________________________________
distribution_lambda_4 (Distr ((None,), (None,))        0
=================================================================
Total params: 12,909
Trainable params: 12,909
Non-trainable params: 0
_________________________________________________________________
```

This depends CPD

Poisson

(None, 1)

Zero Inflated?

(None, 2)

Discretized logistic mixture?

(None, 9)

Validation NLL allows to rank different probabilistic models

Flexible Distributions with Transformation Models

Theory: Name Some Distributions

- Gaussian
- Uniform
- Weibull
- Binomial
- Log-Normal

These are the distributions we have in our Toolbox.

Is the reality like this?

Reality: Data 1-D no covariate (constant x)

What distribution can you use?

Regression Setting

(a) Homoscedastic data, generated by a Gaussian distribution $\mathcal{N}$ with constant variance $\sigma^2 = 2$. The mean is expressed by the linear combination $\mu_x = x + 3$.

Hetero

(b) ~~Homo~~scedastic data, generated by a Gaussian distribution $\mathcal{N}$ with variable variance $\sigma_x^2 = \frac{x}{2}$ and mean $\mu_x = x + 3$.

(c) Bimodal data example derived from a mixture of two Gaussian distribution, with the variable variances $\sigma_{1/2,x}^2 = \frac{x}{2}$ and means $\mu_{1,x} = x + 6$ and $\mu_{2,x} = -1.5 \cdot x + 6$.

Figures: MA Thesis Marcel Arpogaus

Idea of Transformation Models in 1D

Data x~strange_function in $\mathbb{R}^1$

Transformed function $z_1 \sim N(0,1)$

Idea: learn an *invertible* transformation to simple function usually Gaussian $N(0,1)$

- Sampling from $p_x(x)$: sample $z^* \sim p_z(z)$ then transform it via g_θ(z)
- Density of *x**: calculate z*= $g_\theta^{-1}(x^*)$ and evaluate $N(z^*; 0,1)$

There is no fixed naming convention. Also in the direction of g(z) and $g^{-1}(x)$.

Transformation Models Idea

Image credits: Lucas Kook

Transformation Models Simulation

Our deep transformation model

$$f_1: \quad \tilde{y} = a(x) \cdot y - b(x)$$

$$f_2: \quad \tilde{z} = \frac{1}{M+1} \sum_{i=0}^{M} \mathrm{Be}_i\big(\sigma(\tilde{y})\big)\vartheta_i(x)$$

$$f_3: \quad z = \alpha(x) \cdot \tilde{z} - \beta(x)$$

$$h_{\theta(x)} = f_{3,\alpha_x,\beta_x} \circ \; f_{2,\vartheta_{0_x},\ldots,\vartheta_{M_x}} \circ \sigma \circ f_{1,a_x,b_x}$$

$$\mathrm{NLL} = \sum_i -\log\left(f^Z(z_i) \cdot \left| \frac{\partial h_{\widehat{\theta(x_i)}}}{\partial y} \right| \Big|_{y_i} \right)$$

Architecture of our Deep transformation model

$$h_{\theta(x)} = f_{3,\alpha_x,\beta_x} \circ \; f_{2,\vartheta_{0_x},\ldots,\vartheta_{M_x}} \circ \sigma \circ f_{1,a_x,b_x} \qquad \mathrm{NLL} = \sum_i -\log\left(f^z\,(z_i) \cdot \left|\frac{\partial h_{\widehat{\theta(x_i)}}}{\partial y}\right|\Big|_{y_i}\right)$$

Ensuring a monotone increasing h:

Positive slope:

$a > 0$

Increasing Bernstein coefficients :

$\vartheta_0 = \gamma_0$

$\vartheta_1 = \vartheta_0 + e^{\gamma_1}$

...

$\vartheta_M = \vartheta_{M-1} + e^{\gamma_M}$

$\rightarrow \vartheta_0 < \vartheta_1 \ldots < \vartheta_M$

Positive slope:

$\alpha > 0$

Application: Predict CPD for age based on an image

For 10 randomly picked images from test set with **true ages** (1year, 30years, 90 years) the predicted CPD is shown. The solid lines correspond to shown images.

Application: Electric Load Consumption Forecast

(a) Ridge plot for the PLF of Saturday from Figure 50.

Architecture	Probabilistic model	Score
Feedforward	Normal distribution	-9.57
	Gaussian mixture	-90.01
	Normalizing flow	**-98.75**
WaveNet	Normal distribution	-7.65
	Gaussian mixture	-94.14
	Normalizing flow	**-96.09**

Numbers not published yet

Deep Transformation model (further resources)

- R implementation:
  - https://github.com/tensorchiefs/dl_mlt
- Python implementation:
  - https://github.com/MArpogaus/TensorFlow-Probability-Bernstein-Polynomial-Bijector
- Paper
  - https://arxiv.org/abs/2004.00464

Take home messages CPD

- A probabilistic model predicts for each input a whole outcome CPD
- Probabilistic models are the models of choice when dealing with uncertainty
- Probabilistic models fit naturally in DL framework
- Good Probabilistic modeling improves SOTA in deep learning models
- Use the NLL for **training**, **evaluating** and **comparing** probabilistic models

Bayesian Deep Learning

Outline

- Issues with current DL approach
  - No extrapolating / no epistemic uncertainty
- Bayesian Statistics
  - A simple example (Bayes the Hacker's way)
  - Introduction to Bayesian Statistics
- Bayesian Neural Networks
  - Variational Approximation
  - MC-Dropout

Typisches Beispiel einer Kuh

General	VIEW DOCS
cow	0.992
cattle	0.983
mammal	0.979
grass	0.978
livestock	0.966
farm	0.964
landscape	0.963
pasture	0.954
grassland	0.949
agriculture	0.948
no person	0.945

Untypisches Beispiel einer Kuh

Dieses Bild ist zu weit weg von den Trainingsdaten. Kuh nicht unabhängig von Hintergrund

Neuronales Netz erkennt keine Kuh!

General	VIEW DOCS
no person	0.991
beach	0.990
water	0.985
sand	0.981
sea	0.980
travel	0.978
seashore	0.972
summer	0.954
sky	0.946
outdoors	0.944
ocean	0.936

[NeurIPS 2018 Workshop on Causal Learning]

The elephant in the room

Neuronales Netz erkennt keinen Elefanten!

General VIEW DOCS

PREDICTED CONCEPT	PROBABILITY
group	0.979
adult	0.977
people	0.976
furniture	0.960
room	0.957
business	0.903
indoors	0.901
man	0.896
seat	0.895

NeurIPS 2018 Workshop on Causal Learning

Turtle as Rifle

Ausgedruckes Exemplar

Source: https://www.labsix.org/physical-objects-that-fool-neural-nets/

What do we get from a DL classification model?

Suppose you train a classifier to discriminate between 3 dog breeds

The prediction is “collie” because it gets the highest probability: p_{pred}=0.75

What is 0.75 telling us.

What is the probability telling?

p_{pred}=0.75

Among many predications that had p_{pred}=0.75, we expect that on average 75% of these predictions are correct and only 25% predictions are wrong

→ Then the classifier produces *calibrated probabilities*

Sample of images where the predicted class got p=0.75:

Do CNNs produce calibrated probabilities?

Guo et al. (2017)
[On Calibration of Modern NN]

The deeper CNNs get
- the fewer miss-classifications
- the less well calibrated they get

Good news:
deep NN can be “recalibrated” and then we get calibrated probabilities.

CNNs yield high accuracy
and calibrated probabilities, but…

What happens if a novel class is presented to the CNN?

The reported probability is only valid if, we have the P(Train)=P(Test).
That's not the case. "The big lie deep learning is based on".
This is more evident, if we have a look at regression problems.

Elephant in the room

Aufgabe: https://github.com/tensorchiefs/dl_course_2021/blob/master/notebooks/18_elephant_in_the_room.ipynb

Extrapolation

Figure 7.2 Bad case of DL. The high performant VGG16-CNN trained on ImageNet data fails to see the elephant in the room. The five highest-ranked class predictions of the objects in the image are horse_cart, shopping_cart, palace, streetcar, gondola; the elephant is not found! This image is an extrapolation of the training set. In the regression problem on the right side of the dashed vertical line, there's zero uncertainty in the regions where there's no data (extrapolation).

Aleatroic vs. Epistemic Uncertainty

Much spread in data, aleatroic (from “Alea Acta est”))

- *Aleatoric* uncertainty is due to the uncertainty in the data.
- The uncertainty when leaving the ‘known ground’ is called *epistemic* uncertainty.

We can model this uncertainty when we take the uncertainty with which we know the weights (called parameter uncertainty) into account. This can be done with Bayesian reasoning.

Backup

Bayes the hackers' way

Let's look at good old linear regression to understand the gist of the Bayes idea. Assume $\sigma = 3$ to be known.

MaxLike Solution

A bit off the MaxLike Solution

Combining different fits

Also take the other fits with different parameters into account and weight them

Hightest weight for MaxLike

Other other less like solutions, should have a vote.

Question: How to get the weight?

Idea: use the (normalized) likelihood! $p_{norm}((a,b)|D)$ D is training data

Don't put all egg's in one Basket

- Also take other solutions for a,b into account

$$p_{norm}((a,b)|D) = \frac{p((a,b)|D)}{\sum_w p((a,b)|D)}$$

$$\mathrm{p}(y|x,D) = \sum_a \sum_b p(y|x,(a,b)) \cdot p_{norm}((a,b)|D)$$

Likelihood at 30x30 different positions of a and b. Normalized to be one.
https://github.com/tensorchiefs/ dl_book/blob/master/chapter_07 /nb_ch07_02.ipynb

Result

$$\mathrm{p}(y|x,D) = \sum_a \sum_b p(y|x,(a,b)) \cdot p_{norm}((a,b)|D)$$

Figure 7.6 The predictive distribution for the Bayesian linear regression model, trained with the four data points shown on the left side by the color-code and on the right as conditional distribution at two different *x* positions. You can clearly see that the uncertainty gets larger when leaving the *x*-regions where there's data.

What have we done?

- We calculated
  - $\mathrm{p}(y|x,D) = \sum_a \sum_b p(y|x,(a,b)) \cdot p_{norm}((a,b)|D)$

- Or with the weights *w*
  - $\mathrm{p}(y|x,D) = \sum_w p(y|x,w) \cdot p_{norm}\,(w|D)$
  - $\mathrm{p}(y|x,D) = \int_w p(y|x,w) \cdot p_{norm}(w|D)\ dw$ for continuous weights

- The weights now have a distribution, they are random variables as well.

- We choose
  - $p_{norm}(w|D) = \frac{p(w|D)}{\sum_w p(w|D)}$
  - We treated the weights as random variables

  - It turns out that we where not the first to do so…

Bayesian statistics

- The Bayesian Theorem

- Applied to $A = \theta$ and $B = D$
  - Parameters θ of a model e.g. weights *w* of NN
  - Training data *D*

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} = \frac{p(D|\theta)p(\theta)}{\sum_\theta p(D|\theta)p(\theta)} \sim p(D|\theta)p(\theta)$$

– $p(\theta\|D)$	posterior	see next slide
– $p(D\|\theta)$	likelihood	our good old friend
– $p(\theta)$	prior	see next slide
– $p(D)$	evidence	just a normalization constant

The Bayesian Mantra (say it loud)

$$\boldsymbol{p}(\boldsymbol{\theta}|\boldsymbol{D}) \sim \boldsymbol{p}(\boldsymbol{D}|\boldsymbol{\theta})\boldsymbol{p}(\boldsymbol{\theta})$$

"The posterior is proportional to the likelihood, times the prior"

Interpretation updating the degree of belief

- Parameters θ are random variables, with distribution
  - We interpret the distribution as our degree of belief in a certain value

- The Bayes formula is seen as an update of our belief in the light of data

likelihood updates degree of belief

$$p(\theta|D) \sim p(D|\theta)p(\theta)$$

posterior (after seeing data)

prior belief (before seeing data)

Example Coin Toss

- θ parameter for the Bernoulli-distribution (probability of head)
- Before the experiment we assume all value of θ are equally likely $p(\theta) = const$
- Do the experiment, observe 3 times head D='3 heads'
- Calculate likelihood $p(D|\theta) = p(y=1) \cdot p(y=1) \cdot p(y=1) = \theta \cdot \theta \cdot \theta = \theta^3$
- Posterior changes to head

After **Before**

$$p(\theta|D) \sim p(D|\theta)p(\theta)$$

$$\theta^3$$

$p(\theta|D) = 4 \cdot \theta^3$ (normalized so that it integrates to 1)

Posterior Predictive Distribution

From the Hacker's experiment

- $\mathrm{p}(y|x, D) = \int_w p(y|x, \theta) \cdot p(\theta|D)\ dw$

For unconditional (no x)

$$\mathrm{p}(y|D) = \int_\theta p(y|\theta) \cdot p(\theta|D)\ d\theta$$

Prob. of certain result given θ

Posterior prob. of certain value θ

For coin interested in $\mathrm{p}(y = 1|D)$ prob. for head

- $p(y = 1|\theta) = \theta$
- $p(\theta|D) = 4 \cdot \theta^3$

$$\mathrm{p}(y = 1|D) = \int_\theta \theta \ \cdot 4 \cdot \theta^3 d\theta = 0.8$$

Revisiting the Hackers' example

- $\mathrm{p}(y|x, D) = \sum p(y|x, w) \cdot p_{norm}(w|D)$
  - $p_{norm}(w|D) = \frac{p(w|D)}{\sum_w p(w|D)}$
  - Bayes
  - $p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} = \frac{p(D|\theta)p(\theta)}{\sum_\theta p(D|\theta)p(\theta)}$

- What, we did was to set the prior $p(w) = const$
- Graphical Representation, of Bayesian linear regression:

The weights are replaced by distribution. E.g. for a

a --> $p(a|D)$ with $p(a)$ prior

Summary

- $\mathrm{p}(y|x,D) = \int_w p(y|x,w) \cdot p\,(w|D)\ dw$

$p(y|x,D)$

- Not just a single solution
  - "Marginalizing instead of optimization"

- Bayes usually achieves
  - Better uncertainty estimates
  - Better prediction performance