Machine Intelligence:: Deep Learning
Week 1

Beate Sick, Loran Avci, Oliver Dürr

Institut für Datenanalyse und Prozessdesign
Zürcher Hochschule für Angewandte Wissenschaften

Winterthur, 16. March 2021

Outline of the DL Module (tentative)

- Day 1: Jumpstart to DL
  - What is DL
  - Basic Building Blocks
  - Keras
- Day 2: CNN I
  - ImageData
- Day 3: CNN II and RNN
  - Tips and Tricks
  - Modern Architectures
  - 1-D Sequential Data
- Day 4: Looking at details
  - Linear Regression
  - Backpropagation
    - Resnet
  - Likelihood principle
- Day 5: Probabilistic Aspects
  - TensorFlow Probability (TFP)
  - Negative Loss Likelihood NLL
  - Count Data
- Day 6: Probabilistic models in the wild
  - Complex Distributions
  - Generative modes with normalizing flows
- Day 7: Uncertainty in DL
  - Bayesian Modeling
- Day 8: Uncertainty cont'd
  - Bayesian Neural Networks
  - Projects

Projects please register (see also website)
https://docs.google.com/spreadsheets/d/18VFrPbKq3YSOg8Ebc1q1wGgkfgaWl7IkcCCIGEDGj6Q/edit#gid=0

See website for other projects

Learning Objectives for today: looking under the hood

- Get an understanding of
  - Computational Graph
  - Backpropagation in Computational Graph
  - Maximum Likelihood principle for neural networks

Computational Graph

Looking under the hood of tf / Keras

Keras

```python
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

def create_model():
  return tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
  ])
```

TensorFlow

Internal representation (in non-eager mode)
is a computational graph.

https://github.com/tensorflow/tensorboard/blob/master/docs/get_started.ipynb

Next steps

- Understand the computational graph (theoretical)
- Understand backpropagation in a graph theoretical)
- Example Linear Regression (the mother of all networks)

Recap

- The computation in TF is done via a computational graph

- The nodes are ops
- The edges are the flowing tensors

Recap Matrix Multiplication (scalar and with vector)

$$10\begin{pmatrix} 3 & 3 \end{pmatrix}\begin{pmatrix} 2 \\ 2 \end{pmatrix} = 120$$

Be the spider who knits a computational graph

Translate the following TF code in a graph

TensorFlow: Building the graph

```
m1 = tf.constant([[3.0, 3.0]], name='M1')
m2 = tf.constant([[2.0], [2.0]], name = 'M2')
product = 10*tf.matmul(m1,m2)
```

Quite much happen in here!

Finish the computation graph

TensorFlows internal representation

- For fast computation a graph is build
  - Technical detail in tf 2.0 you need to decorate a function with @tf.function to build a graph. Otherwise eager execution happens.

The most important benefit of computational graphs is back propagation…

Motivation: The forward and the backward pass

- https://google-developers.appspot.com/machine-learning/crash-course/backprop-scroll/

Chain rule recap

- If we have two functions f,g
  $y = f(x)\ and$
  $z = g(y)$
  then y and z are dependent variables.

- The chain rule:

$$\frac{\partial z}{\partial x} = \frac{\partial y}{\partial x} * \frac{\partial z}{\partial y}$$

- Backpropagation (flow of the gradient)

$$\frac{\partial z}{\partial x} = \frac{\partial y}{\partial x} * \frac{\partial z}{\partial y}$$

Gradient flow in a computational graph: local junction

$z = f(x, y)$ *and*
$L = f(z)$

Illustration: http://cs231n.stanford.edu/slides/winter1516_lecture4.pdf

Example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$\frac{\partial(\alpha + \beta)}{\partial \alpha} = 1 \qquad \frac{\partial(\alpha * \beta)}{\partial \alpha} = \beta$$

Task (10min): Calculate the derivatives. Once by hand, once with backpropagation (follow the graph)

➔Multiplication do a switch

In depth example: Linear Regression

Example Linear Regression

$$Loss = MSE = 1/n \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = 1/n \sum_{i=1}^{n} (y_i - (a \cdot x_i + b))^2$$

Interlude: Derivations in TF using Tape Mechanism

```
x = tf.Variable(-2.)
y = tf.Variable(5.)
z = tf.Variable(-4.)

with tf.GradientTape() as tape: #We need to store
    res = (x+y)*z
    print(tape.gradient(res, [z]))
```

Forward and Backwardpass for linear regression

Figure 3.12: The forward and backward pass for the concrete example with one data point (x = 58, y = 153) and the initial parameter values a = 0 and b = 139. The flowing values in the forward pass are shown on the left side the graph; the flowing values of the gradients during the backward pass are shown on right side the graph.

Do exercise 10 (works with Tape Gradient)
If you have time you can skim over 11 (Works with the static graph)

Further References / Summary

- For a more in depth treatment have a look at
  - Lecture 4 of http://cs231n.stanford.edu/
  - Slides http://cs231n.stanford.edu/slides/winter1516_lecture4.pdf

- Gradient flow is important for learning: remember!

Consequences of Backprop

Backpropagation through sigmoid

What happens when x = -10?
What happens when x = 0?
What happens when x = 10?

Gradients are killed, when not in active region! Slow learning!

Slide from: CS231

Different activations in inner layers

N-D log regression

$$z = x_1 W_1 + x_2 W_2 + b = Wx + b$$

Activation function a.k.a.
Nonlinearity f(z)

$$f(z) = \begin{cases} \dfrac{\exp(z)}{1+\exp(z)} \\ \max(0,z) \end{cases}$$

Motivation:
Green:
logistic regression.
Red:
ReLU faster
convergence

Figure 1: A four-layer convolutional neural network with ReLUs **(solid line)** reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons

Source:
Alexnet
Krizhevsky et al 2012

There are other alternatives besides sigmoid and ReLU.

Currently ReLU is standard

Backpropagation through ReLU

What happens when x = -10?
What happens when x = 0?
What happens when x = 10?

Gradients are killed, only when x < 0

Slide from: CS231

Recap: Batch Normalization is beneficial in many NN

After BN the input to the activation function is in the sweet spot

Observed distributions of signal after BN before going into the activation layer.

When using BN consider the following:

- Using a higher learning rate might work better
- Use less regularization, e.g. reduce dropout probability
- In the linear transformation the biases can be dropped (step 2 takes care of the shift)
- In case of ReLu only the shift β in steps 2 need to be learned (α can be dropped)

Image credits: Martin Gorner:
https://docs.google.com/presentation/d/e/2PACX-1vRouwj_3cYsmLrNNI3Uq5gv5-hYp_QFdeoan2GlxKgIZRSejozruAbVV0IMXBoPsINB7Jw92vJo2EAM/pub?slide=id.g187d73109b_1_2921

"ResNet" from Microsoft 2015 winner of imageNet

152 layers

ResNet basic design (VGG-style)

- add shortcut connections every two
- all 3x3 conv (almost)

$F(x_l)$

$H(x_l)=x_{l+1} = x_l + F(x_l)$

F(x) is called "residual" since it only learns the "delta" which is needed to add to x to get H(x)

152 layers:
Why does this train at all?

This deep architecture could still be trained, since the gradients can skip layers which diminish the gradient!

credits: https://chatbotslife.com/resnets-highwaynets-and-densenets-oh-my-9bb15918ee32

Closer Look

$$\frac{\partial(\alpha+\beta)}{\partial\alpha}=1$$

➔'Gradient Super Highways'

What comes in (on the right) does go out (on the left)

Similar to LTMS (just in case you know)

Building Loss Functions with Maximum Likelihood

Simple regression via a NN: no probabilistic model in mind

age

Input x

Systolic blood pressure

Output y

$x = 35$ →

→ $\hat{sbp} = 117$

One input x (age) → one predicted outcome (sbp)

Traditional versus probabilistic regression DL models

Describes the spread of the data

Recap Classification: Softmax Activation

$p_o, p_1 \ldots p_9$ are probabilities for the classes 0 to 9.

Activation of last layer z_i incomming

$$p_i = \frac{e^{z_i}}{\sum_{j=0}^{9} e^{z_j}}$$

Makes outcome positive

Ensures that pi's sum up to one

This activation is called softmax

Figure 2.12: A fully connected NN with 2 hidden layers. For the MNIST example, the input layer has 784 values for the 28 x 28 pixels and the output layer out of 10 nodes for the 10 classes.

Neural networks are probabilistic models

- The output of a neural network, can be understood as a probability*
  - Classification
    - Probability of class 1…,K
  - Regression
    - Probability Distribution
- Output of a neural network for training example i

$$p(y_i|x_i, w)$$

Output
- Probability for class y_i
- Distribution

Input

weights

*More on probabilistic interpretation next lecture

Maximum Likelihood

Tune the parameters weights of the network, so that observed data (training data) is most likely.

Practically: Minimize Negative Log-Likelihood of the CPD

$$\hat{w} = argmin \sum_{i=i}^{N} -\log(p(y_i|x_i, w))$$

Maximum Likelihood
(one of the most beautiful ideas in statistics)

Ronald Fisher in 1913
Also used before by
Gauss, Laplace

$$M(\theta) \xrightarrow[\text{(often known)}]{\text{Likelihood / “probability“}} \text{Data}$$

Tune the parameter(s) θ of the model M so that (observed) data is most likely

What‘s the likelihood of the data for lin. regression...

Motivating Example of MaxLike

Figure 4.2 A die with one side showing a dollar sign and the others a dot.

Question: What is probability to see one $-signs in 10 throws?

A see Blackboard:

k ~ binom(n=10, p = 1/6)

Solution

```
from scipy.stats import binom
ndollar = np.asarray(np.linspace(0,10,11), dtype='int')
pdollar_sign = binom.pmf(k=ndollar, n=10, p=1/6)
plt.stem(ndollar, pdollar_sign)
plt.xlabel('Number of dollar signs’)
plt.ylabel('Probability')
```

Excercise

Now you don’t know how many dollar signs are on the die.

You throw the die 10 times and get k=2 dollar signs.

What is you best guess?

Work Through Exercise:

Work through the code until you reach the first exercise. In the exercise it is your task to determine the probability to observe two-times a dollar sign in ten dice throws, if you consider a die that has dollar signs on 0, 1, 2, 3, 4, 5, or all 6 faces.

https://github.com/tensorchiefs/dl_book/blob/master/chapter_04/nb_ch04_01.ipynb

ML principle for binary classification

x_i, y_i Training data $i = 1, \dots N$

$p_0(x_i)$ is probability for $y_i = 0$

$p_1(x_i)$ is probability for $y_i = 1$

Question:
What is probability for the training set of say 5 examples? First 3 of class 0, last two 2 of class 1?

ML principle for binary classification

x_i, y_i Training data $i = 1, \ldots N$

$p_0(x_i)$ is probability for $y_i = 0$

$p_0(x_i)$ is probability for $y_i = 1$

Answer:
What is probability for the training set of say 5 examples? First 3 of class 0, last two 2 of class 1?

$$Pr(Training) = p_0(x_1) \cdot p_0(x_2) \cdot p_0(x_3) \cdot p_1(x_4) \cdot p_1(x_5) = \prod_{j=1}^{3} p_0(x_j) \cdot \prod_{j=4}^{5} p_1(x_j)$$

Taking the log

To determine the maximal value, taking log is also ok.

Negative Log-Likelihood (NLL)

- Likelihood of training data

$$Pr(Training) = \prod_{j\ for\ with\ y=0} p_0(x_j) \cdot \prod_{j\ for\ with\ y=1} p_1(x_j)$$

- LogLike

$$\log(Pr(Training)) = \sum_{j\ for\ y=0} \log(p_0(x_j)) + \sum_{j\ for\ y=1} \log(p_1(x_j))$$

- Crossentropy / NNL negative log likelihood (per example divided by n)

$$crossentropy = -\frac{1}{n}\left(\sum_{j\ for\ y=0} \log(p_0(x_j)) + \sum_{j\ for\ y=1} \log(p_1(x_j))\right)$$

More than 2 classes

Figure 2.11 The first four digits of the MNIST data set—the standard data set used for benchmarking NN for images classification

$$crossentropy = -\frac{1}{n}\left(\sum_{j\ for\ y=0} \log(p_0(x_j)) + \sum_{j\ for\ y=1} \log(p_1(x_j)) + \ldots + \sum_{j\ for\ y=K-1} \log(p_{k-1}(x_j))\right)$$

$$crossentropy = -\frac{1}{n}\sum_{i=1}^{n} {}^{true}p_i \cdot log(p_i)$$

Excercise

Task:

- Calculate (with calculator or numpy) the excepted cross-entropy for the MNIST example if you just guess, each class with p=1/10.

- Load MNIST and make a small CNN without training and calculate the loss

- See NB 12b_mnist_loglike

Backup
Loss function for regression

Regression

Figure 4.12 Scatterplot and regression model for the blood pressure example: The dots are the measured data points, the straight line is the linear model, the bell-shaped curves are the conditional probability distributions of the outcome Y conditioned on the observed value x. The length of the red bar indicates likelihood for the observed sbp of 131 for a 22 year old woman.

Zoom for one example

$$f(y = 131, \mu = 111, \sigma = 20) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-\mu)^2}{2\sigma^2}} = \frac{1}{\sqrt{2\pi \cdot 400}} e^{-\frac{(131-111)^2}{2400}} \approx 0.01209$$

Figure 4.13 The conditional normal density function f. The height of the red bar indicates the likelihood of the specific value under this model.

For all training Examples:

$$L(a,b) = f(y_1, \mu_{x_2}, \sigma) \cdot f(y_2, \mu_{x_2}, \sigma) \ldots = \prod_{i=1}^{n} f(y_i, \mu_i, \sigma) = \prod_{i=1}^{n} f(y_i, a \cdot x_i + b)$$

MSE-Loss

$$w = argmax_w \left\{ \prod_{i=1}^{n} f(y_i|x_i, w) \right\} \Rightarrow$$

$$w = argmin_w \left\{ \sum_{i=1}^{n} - log\, f(y_i|x_i, w) \right\} \Rightarrow$$

$$w = argmin_w \left\{ \sum_{i=1}^{n} - log\left(\frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(y_i-\mu_{x_i})^2}{2\sigma^2}}\right) \right\} \Rightarrow$$

$$w = argmin_w \left\{ \sum_{i=1}^{n} - log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) + \frac{(\mu_{x_i}-y_i)^2}{2\sigma^2} \right\} \quad (5) \Rightarrow$$

Here we assume constant spread

$$w = argmin_w \left\{ \sum_{i=1}^{n} + \frac{(\mu_{x_i}-y_i)^2}{2\sigma^2} \right\} \Rightarrow$$

$$w = argmin_w \left\{ \frac{1}{2\sigma^2} \sum_{i=1}^{n} (\mu_{x_i} - y_i)^2 \right\} \Rightarrow$$

$$(a, b) = argmin_w \left\{ \frac{1}{n} \sum_{i=1}^{n} (\mu_{x_i} - y_i)^2 \right\}$$