Deep Learning for Natural Language Processing

Lecture 2 – Machine Learning Basics

Dr. Ivan Habernal

April 20, 2021

Trustworthy Human Language Technologies
Department of Computer Science
Technical University of Darmstadt

www.trusthlt.org

This lecture

Machine Learning Principles

- Train/dev/test split
- Evaluation
- Loss functions

Learning goals

- Understand ML/DL foundations

Notation

Vectors in linear algebra are columns, for example $\mathbf{x} \in \mathbb{R}^3$

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \qquad \text{(bold face, lower case)}$$

We treat them as a row vector by transposing, for example $\mathbf{x}^\intercal = (x_1, x_2, x_3)$ — which is a matrix $\mathbb{R}^{1\times 3}$

Caveat: 1-D array (a list of numbers) is sometimes considered a vector, so dealing with dimensions might be quite messy

Notation

Matrices are upper-case bold, for example $\mathbf{Z} \in \mathbb{R}^{2\times 3}$

$$\mathbf{Z} = \begin{pmatrix} z_{1,1} & z_{1,2} & z_{1,3} \\ z_{2,1} & z_{2,2} & z_{2,3} \end{pmatrix}$$

Scalars are ordinary lower case letters, for example

$$a, b, c \in \mathbb{R}$$

Notation ambiguity

A dot $\cdot$ means multiple things, depending on context

Simple scalar multiplication, for example $a \cdot b$

$$\cdot : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$$

Dot product $\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{n} x_i y_i$

$$\cdot : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$$

Matrix-matrix (matrix-vector/vector-matrix) multiplication, for example $\mathbf{x} \cdot \mathbf{W}$ or $\mathbf{Y} \cdot \mathbf{Z}$

$$\cdot : \mathbb{R}^{m \times n} \times \mathbb{R}^{n \times p} \to \mathbb{R}^{m \times p}$$

Derivatives

Derivative of function $f(x)$ is denoted as $\frac{\mathrm{d}f}{\mathrm{d}x}$ (rarely as f')

Partial derivates of a function of several real variables $f(x_1, \ldots, x_n)$ we use

$$\frac{\partial f}{\partial x_i}$$

The gradient is then a row vector of partial derivatives

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right)$$

Notation

Vector-matrix multiplication

- What is $\mathbf{x} \cdot \mathbf{W}$ if $\mathbf{x} \in \mathbb{R}^{1 \times n}$ and $\mathbf{W} \in \mathbb{R}^{n \times d}$?

Cosine similarity

- For two vectors $\mathbf{x}, \mathbf{y}$, their *cosine similarity* is defined as

$$\frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|} = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2} \cdot \sqrt{\sum_i y_i^2}} \qquad \in [0, 1]$$

Functions applied element-wise to vectors, e.g.,
$f : \mathbb{R}^{1 \times n} \to \mathbb{R}^{1 \times n}$

$$f(\mathbf{x}) = (f(x_i), \ldots, f(x_n))$$

Supervised Machine Learning basics

Problem setup

We have N labeled **data points** (or examples) as tuples

$$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)$$

where y_n is the "truth" or "gold label" of $\mathbf{x}_n$.

We have a **model** parametrized by θ that outputs y (or $\hat{y}$)

$$\hat{y} = f_\theta(\mathbf{x})$$

We specify a **loss function**, for example

$$\frac{1}{N} \sum_{i=1}^{N} \left(y_i - f_\theta(\mathbf{x}_i)\right)^2$$

Learning

Our goal is to find such parameters θ that minimize the loss

- In other words, our model ”fits” the training data ”better”

Overfitting

If our model is sufficiently ”rich” (huge number of parameters), it could minimize the loss by **remembering** our training data perfectly $\rightarrow$ Not the goal of learning!

Generalization

The actual goal of machine learning is to **generalize** well on previously unseen data

Evaluating generalization?

- Split dataset into training and test

- Models must perform well on test data (hidden during learning)

Regularization and hyperparameters

Regularization for preventing overfitting by putting constraints on θ (e.g., penalizing large parameter values)

Hyperparameters?

- Learning rate, early stopping, batch size, etc.

- Hyperparameter tuning: Split training data into training and development

ML Basics

Three major components of a machine learning system

- Data,
- Models
- Learning

Supervised learning problem: Data

Dataset is a set of input-label tuples (labeled examples)

$$\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n), \ldots, (\mathbf{x}_N, y_N)\}$$

- N denotes the number of examples in a dataset, we index the examples with lowercase $n = 1, \ldots, N$.

- Each input $\mathbf{x}_n$ is a D-dimensional vector of real numbers, which are called features, attributes, or covariates

- Label y_n associated with input vector $\mathbf{x}_n$

Compact representation of the dataset inputs: $\mathbf{X} \in \mathbb{R}^{N \times D}$

Models as Functions

Predictor is a function that, when given a particular input example produces an output

$$f : \mathbb{R}^D \rightarrow \mathbb{R}$$

- For illustration, we predict single real (regression)

- In classification we typically predict a probability distribution over categories, e.g.

$$f : \mathbb{R}^D \rightarrow \mathbb{R}^{|C|}$$

where $|C|$ is the number of classes, and the (arbitrary) mapping is

Models as Functions

Classification models typically predict a probability distribution over categories ($|C|$ is the number of classes)

$$f : \mathbb{R}^D \rightarrow \mathbb{R}^{|C|}$$

For example

$$C = \begin{cases} 0 & \text{Sport} \\ 1 & \text{Politics} \\ 2 & \text{Business} \end{cases}$$

$$f : \mathbb{R}^D \rightarrow \underbrace{(0.01, 0.82, 0.17)}_{\sum=1.0}$$

Learning is Finding Parameters

The goal of learning is to

- find a model and its corresponding parameters
- such that the resulting predictor will perform well on unseen data

Conceptually three distinct algorithmic phases when discussing machine learning algorithms

1. Prediction or inference
2. Training or parameter estimation
3. Hyperparameter tuning or model selection

Hypothesis Class of Functions

Supervised learning on dataset $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$; $x_n \in \mathbb{R}^D$

Estimate a predictor parametrized by θ (e.g., a vector of $\mathbb{R}$ parameters)

$$f(\cdot, \theta) : \mathbb{R}^D \to \mathbb{R}$$

We hope to "find" "good" parameters θ^* so that we "fit" the data well

$$f(\mathbf{x}_n, \theta^*) \approx y_n \qquad \text{for all } n = 1, \ldots, N$$

Notation: let $\hat{y_n} = f(\mathbf{x}_n, \theta^*)$ represent predictor's output

Loss Function for Training

What does it mean to fit the data "well"?

We need to specify a **loss function**

$$\ell(\underbrace{y_n}_{\text{True label}}, \underbrace{\hat{y}_n}_{\text{Predictor's output}}) \rightarrow \underbrace{\mathbb{R}^+}_{\text{"Loss"}}$$

- representing how much error we have made on this particular prediction

Our goal for finding a good parameter vector θ^* is to **minimize the average loss** on the set of N training examples

Independent and identically distributed

Assumption: Our dataset $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ is **Independent and identically distributed (I.I.D)**

- Two data points $(\mathbf{x}_i, y_i)$ and $(\mathbf{x}_j, y_j)$ do not statistically depend on each other

- Implication: We can use the **empirical mean** of the loss on the training data ("empirical risk")

$$\mathbf{R}_{\text{emp}}(f, \mathbf{X}, \mathbf{y}) = \frac{1}{N}\left[\ell(y_1, \hat{y_1}) + \cdots + \ell(y_N, \hat{y_N})\right] = \frac{1}{N}\sum_{i=1}^{N} \ell(y_i, \hat{y}_i)$$

Loss example: Squared Loss

$$\ell(y_n, \hat{y_n}) = (y_n - \hat{y_n})^2$$

Minimizing empirical risk

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} (y_i - f(\mathbf{x}, \theta))^2$$

Expected risk

Not interested in a predictor that only performs well on the training data

We seek a predictor that performs well (has low risk) on unseen test data.

That is, finding a predictor f (with parameters fixed) that minimizes the **expected risk**

$$\mathbf{R}_{\text{true}}(f) = \mathbb{E}_{\mathbf{x},y}\left[\ell(y, f(\mathbf{x}))\right]$$

y is the label; $f(\mathbf{x})$ is the prediction based on example x

The expectation is over the (infinite) set of all possible data and labels

Expected risk

Minimizing the **expected risk**

$$\mathbf{R}_{\text{true}}(f) = \mathbb{E}_{(\mathbf{x},y)\sim P}\left[\ell(y, f(\mathbf{x}))\right] = \int_{\mathcal{X}\times\mathcal{Y}} \ell(y, f(\mathbf{x}))P(\mathbf{x}, y)\mathrm{d}\mathbf{x}\mathrm{d}y$$

The expectation is over the (infinite) set of all possible data and labels.

- How to estimate expected risk from (finite) data?
- How to change training to generalize well?

Empirical risk minimization = Approximately minimizing expected risk

Simulate unseen data: hold out a proportion of the whole dataset

Generalization

We’re interested in generalization performance, not how predictor works on training data

We always split our data:

- The **training set** is used to fit the model

- The **test set** is used to evaluate generalization performance

Test set not seen by the machine learning algorithm during training

Overfitting

Empirical risk minimization can lead to **overfitting**

The predictor fits too closely to the training data and does not generalize well to new data

- Having very small average loss on the training set but large average loss on the test set?

- Tends to occur when we have little data and a complex hypothesis class

Overfitting

Predictor f (with parameters fixed); overfitting occurs when the risk estimate from the training data

$$\mathbf{R}_{\text{emp}}(f, \mathbf{X}_{\text{train}}, \mathbf{y}_{\text{train}})$$

underestimates the expected risk $\mathbf{R}_{\text{true}}(f)$.

Since we estimate the expected risk $\mathbf{R}_{\text{true}}(f)$ by using the empirical risk on the test set

$$\mathbf{R}_{\text{emp}}(f, \mathbf{X}_{\text{test}}, \mathbf{y}_{\text{test}})$$

if the test risk is much larger than the training risk, this is an indication of overfitting.

Cross-validation

K-fold cross-validation partitions the data into K chunks

$K - 1$ of which form the training set $\mathcal{R}$

The last chunk serves as the validation set $\mathcal{V}$

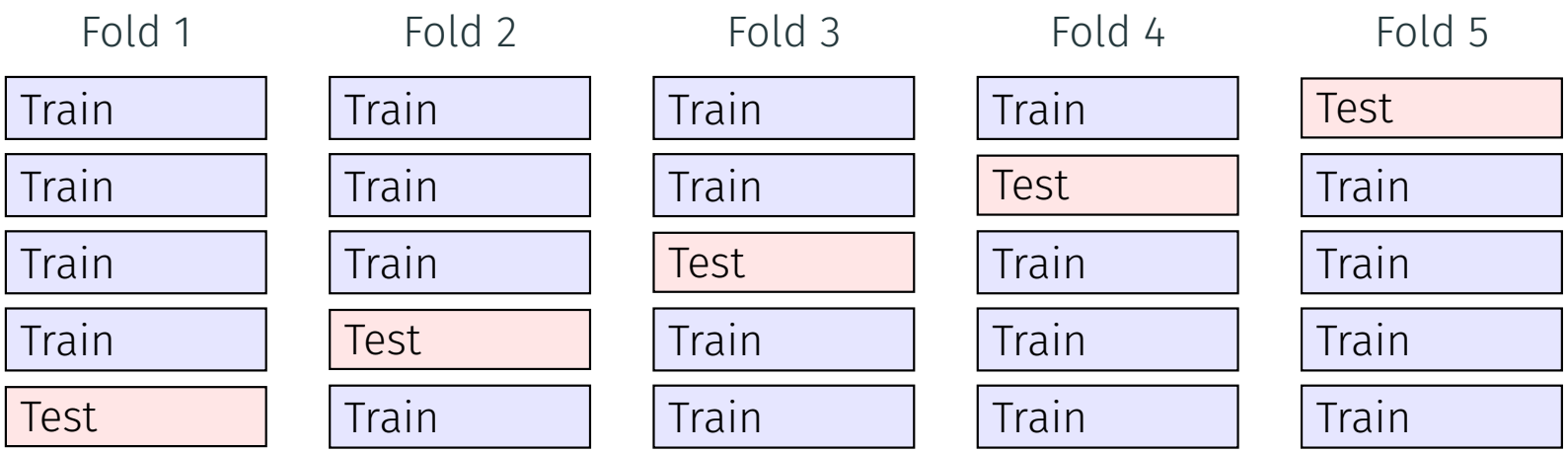

Figure 1: Example of 5-fold CV

Cross-validation

For each partition k, the training data $\mathcal{R}^{(k)}$ produces a predictor $f^{(k)}$

It is then applied to validation set $\mathcal{V}^{(k)}$ to compute the empirical risk $\mathbf{R}(f^{(k)}, \mathcal{V}^{(k)})$

Cross-validation approximates the expected generalization error

$$\mathbb{E}_{\mathcal{V}}\left[\mathbf{R}(f, \mathcal{V})\right] \approx \frac{1}{K} \sum_{k=1}^{K} \mathbf{R}(f^{(k)}, \mathcal{V}^{(k)})$$

Hyper-parameter optimization

Training data can be also split to two parts: Training and Development set

Development set used for optimizing hyper-parameters

Nested cross-validation

- For model selection

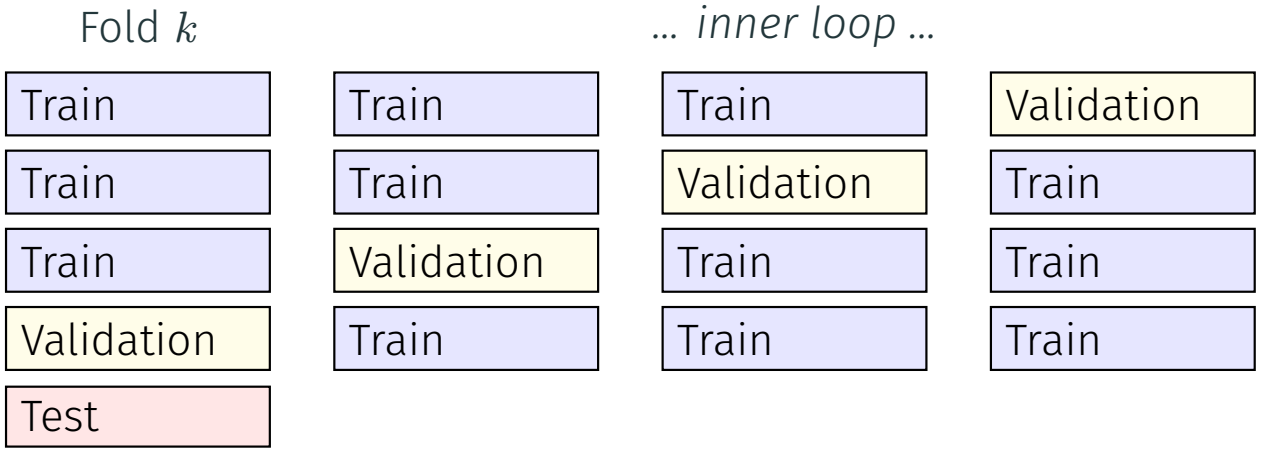

Figure 2: Example of 5-fold nested CV

Evaluation

We often use **accuracy** on the test data to evaluate our model performance:

- How many instances are correctly classified divided by the number of instances in the test set

- However, there are other plausible evaluation measures:

When your outputs are continuous:

- Squared distance (MSE), cosine, correlation, etc.

Say, your output is a sequence:

- Could use edit distance, for instance

Evaluation

Two systems predict whether a patient has a rare disease Q. Which system is better?

	Prediction is Q	Prediction is not Q
Patient has Q	0	10
Patient has not Q	1	1004

Table 1: System A

	Prediction is Q	Prediction is not Q
Patient has Q	4	6
Patient has not Q	5	1000

Table 2: System B

Both systems have accuracy 1004/1015 = 0.99

Terminology

Confusion matrix for binary classification

	Prediction is Q	Prediction is not Q
Truth is Q	True positive (**TP**)	False negative (**FN**)
Truth is not Q	False positive (**FP**)	True negative (**TN**)

Precision and recall

For Class *Disease*

- System B has **precision** of $4/9 = 0.444$
- and recall of $4/10 = 40$
- System A has precision of 0 and recall of 0

For Class *No Disease*

- both systems are very close:

- Precision A: 1004/1014, Pr B: 1000/1006
- Recall A: 1004/1005, Rec B: 1000/1005

F-measure

When there are more than two classes, precision and recall for class k are defined as

$$\mathrm{P}_k = \frac{C_{k,k}}{\sum_i C_{i,k}} \qquad \mathrm{R}_k = \frac{C_{k,k}}{\sum_i C_{k,i}}$$

where C is a confusion matrix as above

From precision and recall, compute the F_1-score

$$F_1 = \frac{2\mathrm{PR}}{\mathrm{P} + \mathrm{R}}$$

Evaluation – F1 measure

For two or more classes, we typically compute the F1-score of each class and then combine this in an overall score:

- For example, averaging all the F1 scores
- There are several ways of averaging with different names (e.g. micro F1 vs. macro F1)

Very relevant reading:

G. Forman and M. Scholz (2010). "Apples-to-Apples in Cross-Validation Studies: Pitfalls in Classifier Performance Measurement". In: *ACM SIGKDD Explorations Newsletter* 12.1, pp. 49–57

M. Sokolova and G. Lapalme (2009). "A systematic analysis of performance measures for classification tasks". In: *Information Processing and Management* 45.4, pp. 427–437. URL: http://dx.doi.org/10.1016/j.ipm.2009.03.002

Other evaluation metrics in NLP

Evaluating **language generation** is not trivial[1]

- Machine Translation (MT)
  - BLEU is based on n-gram matches between the candidate and reference sentences
    - but increase in BLEU does not always indicate an improvement in quality
- Summarization
  - ROUGE-L: longest common sub-sequences between the candidate and references, i.e. a set of shared words with similar order even if not contiguous

[1] O. Caglayan, P. Madhyastha, and L. Specia (2020). "Curious Case of Language Generation Evaluation Metrics: A Cautionary Tale". In: *Proceedings of COLING*, pp. 2322–2328

Loss functions

Loss over multi-dimensional output

We already introduced loss function as $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}^+$

$$\ell(\underbrace{y_n}_{\text{True label}}, \underbrace{\hat{y_n}}_{\text{Predictor's output}}) \to \underbrace{\mathbb{R}^+}_{\text{"Loss"}}$$

For a neural network with multi-dimensional output (M output neurons), we extend the loss to $\ell : \mathbb{R}^M \times \mathbb{R}^M \to \mathbb{R}^+$

$$\ell(\underbrace{\mathbf{y}_n}_{\text{True label}}, \underbrace{\hat{\mathbf{y}}_n}_{\text{Predictor's output}}) \to \underbrace{\mathbb{R}^+}_{\text{"Loss"}}$$

Where $\mathbf{y}_n = (y_{n,1}, y_{n,2}, \ldots, y_{n,m}, \ldots, y_{n,M})$ or if n is clear from context simply $\mathbf{y} = (y_1, y_2, \ldots, y_m, \ldots, y_M)$

Multi-dimensional square loss

For $M = 1$, we had[2]

$$\ell(y, \hat{y}) = (y - \hat{y})^2$$

For multiple dimensions

$$\ell(\mathbf{y}, \hat{\mathbf{y}}) = (\mathbf{y}_m - \hat{\mathbf{y}}_m)^2 = \sum_{m=1}^{M} (y_m - \hat{y_m})^2$$

[2] For each example n; here dropping out n from the formula for clarity

Background: K-L divergence

Also known as *relative entropy*

Let Y and $\hat{Y}$ be categorical random variables over same categories, with probability distributions $P(Y)$ and $Q(\hat{Y})$

$$
\begin{aligned}
\mathbb{D}(P(Y)||Q(\hat{Y})) &= \mathbb{E}_{P(Y)}\left[\log \frac{P(Y)}{P(\hat{Y})}\right] \\
&= \mathbb{E}_{P(Y)}\left[\log P(Y) - \log P(\hat{Y})\right] \\
&= \mathbb{E}_{P(Y)}\left[\log P(Y)\right] - \mathbb{E}_{P(Y)}\left[\log P(\hat{Y})\right] \\
&= -\mathbb{E}_{P(Y)}\left[\log \frac{1}{P(Y)}\right] - \mathbb{E}_{P(Y)}\left[\log P(\hat{Y})\right] \\
&= -\mathbb{H}_{P}(Y) - \mathbb{E}_{P(Y)}\left[\log P(\hat{Y})\right]
\end{aligned}
$$

Cross-entropy loss

Labels as categorical probability distributions

$$\mathbb{D}(P(Y)||Q(\hat{Y})) = -\mathbb{H}_P(Y) - \mathbb{E}_{P(Y)}\left[\log P(\hat{Y})\right]$$

$\mathbb{H}_P(Y)$ does not depend on predictions $\hat{Y}$ so we only care about $-\mathbb{E}_{P(Y)}\left[\log P(\hat{Y})\right]$; Let $P(Y = m) = y_m$

$$\ell(\mathbf{y}, \hat{\mathbf{y}}) = -\mathbb{E}_{P(Y)}\left[\log P(\hat{Y})\right] = -\sum_{m=1}^{M} P(Y = m) \log P(\hat{Y} = m)$$

$$= -\sum_{m=1}^{M} y_m \log(\hat{y}_m)$$

Examples

$\mathbf{y}_n$ is n-th example true label, $\hat{\mathbf{y}}_n$ is n-th example prediction

$$\mathbf{y}_n = (0, 1, 0, 0) \qquad \hat{\mathbf{y}}_n = (0.25, 0.3, 0.4, 0.05)$$

Cross-entropy loss $\ell(\mathbf{y}, \hat{\mathbf{y}}) = \sum_m y_m \log(\hat{y}_m)$

$\ell(\mathbf{y}, \hat{\mathbf{y}}) = -\log 0.3 \approx 1.737$

Square loss $\ell(\mathbf{y}, \hat{\mathbf{y}}) = \sum_m (y_m - \hat{y_m})^2$

$\ell(\mathbf{y}, \hat{\mathbf{y}}) = (0 - 0.25)^2 + (1 - 0.3)^2 + (0 - 0.4)^2 + (0 - 0.05)^2$

Comparison of square loss and cross-entropy loss

$$\mathbf{y}_n = (1, 0) \qquad \hat{\mathbf{y}}_n = (z, 1 - z) \qquad z \in \mathbb{R} : \langle 0, 1 \rangle$$

Square loss

$$\ell(\mathbf{y}_n, \hat{\mathbf{y}_n}) = 2(1 - z)^2$$

Cross-entropy loss

$$\ell(\mathbf{y}_n, \hat{\mathbf{y}_n}) = -\log z$$

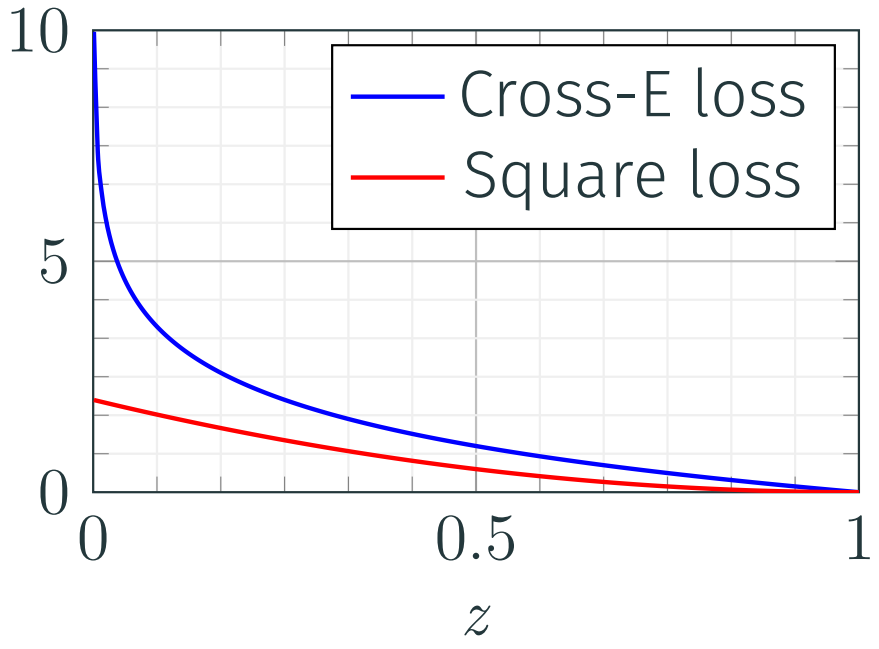

Figure 3: Comparison of losses as function of z for a concrete example

Summary

”Bread & butter” of machine learning

- Notation
- Goals of supervised machine learning
- Scenarios for training and testing, cross-validation
- Evaluation
- Loss functions

Further suggested reading

Chapter 8 of M. P. Deisenroth, A. Faisal, and C. S. Ong (2021). *Mathematics for Machine Learning*. Cambridge University Press. URL: `mml-book.com`

Chapter 5 of I. Goodfellow, Y. Bengio, and A. Courville (2016). *Deep Learning*. MIT Press. URL: `www.deeplearningbook.org`

For probabilistic treatment Chapter 16 of D. Koller and N. Friedman (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press