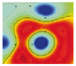
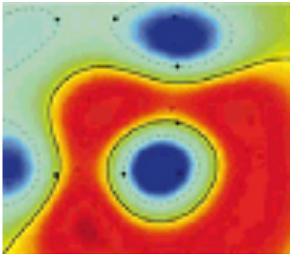


WiSe 2024/25

**Deep Learning 1**

Technische Universität Berlin



# Lecture 7 | **Loss Functions**

## Outline

**Recap: Formulating the learning problem**

**Loss functions for regression**

- 0/1 loss, squared loss, absolute loss, logcosh
- Incorporating predictive uncertainty

**Loss functions for classification**

- 0/1 loss, perceptron loss, log loss
- Extensions to multiple classes

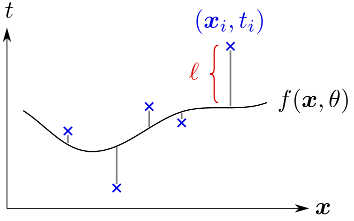
**Practical Aspects**

- Utility-based loss functions
- Incorporating data quality
- Multiple tasks

## Formulating the Learning Problem

Objective to minimize is often defined as the average over the training data of a *loss function* $\ell$, measuring for each instance $i$ the discrepancy between the prediction $y_i = f(\boldsymbol{x}_i, \theta)$ and the ground-truth $t_i$.

$$\mathcal{E}(\theta) \quad \frac{1}{N} \sum_{i=1}^{N} \ell(y_i, t_i)$$



**Two factors influence the learned model $f$:**

- What data is available for training the model (Lectures 5 and 6).
- The choice of loss function, e.g. whether larger errors are penalized more than small errors (today's lecture).

Part 1

## Loss Functions for Regression

## Regression Losses

**Observations:**

- In numerous applications, one needs to predict real values (e.g. age of an organism, expected durability of a component, energy of a physical system, value of a good, product of a chemical reaction, yield of a machine, temperature next week, etc.).
- For these applications, labels are provided as real-valued targets $t \in \mathbb{R}$, and one needs to choose a loss function that quantifies well the difference between such target value and the prediction $f(\boldsymbol{x}) \in \mathbb{R}$.

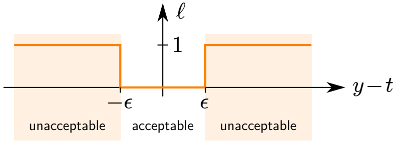
**Several considerations for designing $\ell$:**

- What is the cost of making certain types of errors? Are small errors tolerated? Are big errors more costly?
- What is the quality of the ground-truth target values in the dataset? Are there some outliers?

## The 0/1 Loss

**Function to minimize:**

$$\ell(y,t) = \begin{cases} 0 & -\epsilon \leq y - t \leq \epsilon \\ 1 & \text{else} \end{cases}$$



**Advantages:**

- Tolerant to some small task-irrelevant discrepancies ($\rightarrow$ does not need to fit the data exactly) and can therefore accomodate simple, better-generalizing models.
- Not affected by potential outliers in the data (just treat them as regular errors).

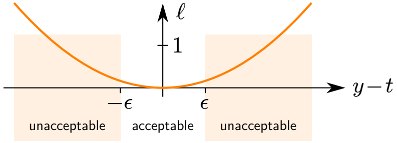
**Disadvantage:**

- The gradient of that loss function is almost always zero $\rightarrow$ impossible to optimize via gradient descent.

## The Squared Loss

**Function to minimize:**

$$\ell(y,t) = (y-t)^2$$



**Advantages:**

- Tolerant to some small task-irrelevant discrepancies.
- Unlike the 0/1 loss, gradients are most of the time non-zero. This makes this loss easy to optimize.

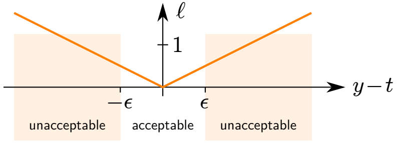
**Disadvantage:**

- Strongly affected by outliers (errors grow quadratically).

## The Absolute Loss

**Function to minimize:**

$$\ell(y,t) = |y - t|$$



**Advantages:**

- Compared to the square error, less affected by outliers (errors grow only linearly).
- Non-zero gradients $\rightarrow$ easy to optimize.

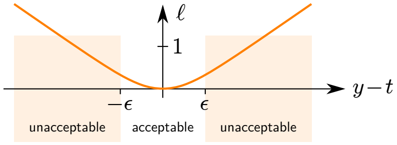
**Disadvantage:**

- Unlike the 0/1 loss and the square error, it is not tolerant to small errors (small errors incur a non-negligible cost).

## The Log-Cosh Loss

**Function to minimize:**

$$\ell(y,t) = \frac{1}{\beta} \log\cosh(\beta \cdot (y-t))$$



with $\beta$ a positive-valued hyperparameter.

**Advantages:**

- Tolerant to some small task-irrelevant discrepancies.
- Non-zero gradients everywhere (except when the prediction is correct). This makes this loss easy to optimize.
- Only mildly affected by outliers (error grows linearly).

## Regression Losses

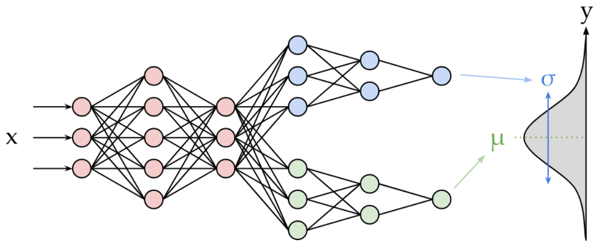
**Systematic comparison**

| | optimizable | outlier-robust | $\epsilon$-tolerant |
|---|---|---|---|
| 0/1 loss | ✗ | ✓ | ✓ |
| squared loss $(y-t)^2$ | ✓ | ✗ | ✓ |
| absolute loss $\lvert y-t \rvert$ | ✓ | ✓ | ✗ |
| log-cosh loss | ✓ | ✓ | ✓ |

**Note:**

- Many further loss functions have been proposed in the literature (e.g. Huber's loss, $\epsilon$-sensitive loss, etc.). They often implement similar desirable properties as the log-cosh loss.

## Regression Losses: Adding Predictive Uncertainty



**Idea:**

- Let the network output consist of two variables $\mu, \sigma$, representing the parameters of some probability distribution modeling the labels $t$, for example a normal distribution $y \sim \mathcal{N}(\mu, \sigma)$.
- We can then define the log-likelihood function, which we would like to maximize w.r.t. the parameters of the network:

$$\log p(y = t \,|\, \mu, \sigma) = -\frac{(t-\mu)^2}{2\sigma^2} - \log(\sqrt{2\pi}\sigma)$$

## Regression Losses: Adding Predictive Uncertainty

Objective to maximize:

$$\log p(y = t \,|\, \mu, \sigma) = -\frac{(t-\mu)^2}{2\sigma^2} - \log(\sqrt{2\pi}\sigma)$$

**Observation:**

- The objective has a gradient w.r.t. $\mu$ and $\sigma$ (as long as the scale $\sigma$ is positive and not too small). To ensure this, one can use some special activation function to produce $\sigma$, e.g. $\sigma = \log(1+\exp(\cdot))$.
- If we set $\sigma$ constant (i.e. disconnect it from the rest of the network), the model reduces to an application of the square error loss function. However, if we learn $\sigma$, the latter provides us with a an indication of prediction uncertainty.
- If we choose different data distributions, we recover different loss functions (e.g. the Laplace distribution yields the absolute loss, or the hyperbolic secant distribution yields the log-cosh loss).

Part 2

## Loss Functions for Classification

## Classification Losses

**Observations:**

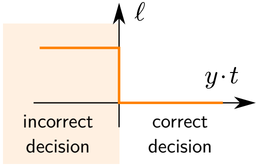
- Classification is perhaps the most common scenario in machine learning (e.g. detecting if some tissue contains cancerous tissue or not, determining whether to grant access or not to some resource, detecting if some text is positive or negative, etc.)
- For these applications, labels are provided as elements of a set, typically $t \in \{-1, 1\}$ for binary classification or $t \in \{1, 2, \dots, C\}$ for multi-class classification.
- However, the output of the neural network is, like for the regression case, real-valued. For binary classification, it is typically a real-valued scalar the sign of which gives the class. The classification is then correct if and only if:

$$\big((y > 0) \wedge (t = 1)\big) \vee \big((y < 0) \wedge (t = -1)\big)$$

and this can be written more compactly as:

$$y \cdot t > 0$$

## 0/1 Loss



**Function to minimize:**

$$\ell(y,t) = \begin{cases} 0 & \text{if} \quad y \cdot t > 0 \\ 1 & \text{if} \quad y \cdot t < 0 \end{cases}$$

**Properties:**

- Using the 0/1 loss function is equivalent to minimizing the average classification error on the training data.
- If the training data would exactly correspond to the test distribution, then, the optimization objective would exactly maximize what we are interested in, i.e. the classification accuracy.

**Problem:**

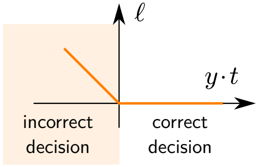
- The loss function has gradient zero everywhere $\Rightarrow$ It can't be optimized via gradient descent.

## Perceptron Loss

**Function to minimize:**

$$\ell(y,t) = \begin{cases} 0 & \text{if} \quad y \cdot t > 0 \\ |y| & \text{if} \quad y \cdot t < 0 \end{cases}$$

Note that it can also be formulated more compactly as $\ell(y,t) = \max(0, -y \cdot t)$.

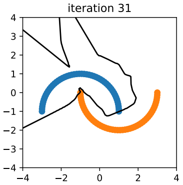


**Advantage:**

- Gradient is non-zero for misclassifications and indicates how to adapt the model to reduce the classification errors.
- Remains fairly capable of dealing with misclassified data (like the 0/1 loss), because the error only grows linearly with $y$.

**Disadvantage:**

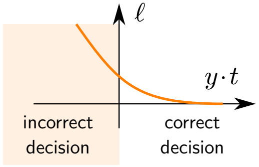
- Training stops as soon as training points are on the correct side of the decision boundary. $\rightarrow$ Unlikely to generalize well to new data points (the 0/1 loss function has the same problem).



## Log-Loss

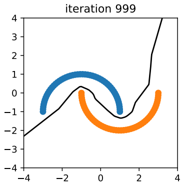
**Function to minimize:**

$$\ell(y,t) = \log(1 + \exp(-y \cdot t))$$



**Advantages:**

- Penalize points that are correctly classified if the neural network output is too close to the threshold. This pushes the decision boundary away from the training data and provide intrinsic regularization properties.



## Log-Loss

**Probabilistic interpretation:**

Assuming the following mapping from neural network output $y$ to class probabilities

$$p = \Big(\frac{\exp(-y)}{1+\exp(-y)}, \frac{\exp(y)}{1+\exp(y)}\Big)$$

minimizing the log-loss is equivalent to minimizing the cross-entropy $H(q,p)$ where $q = (1_{t<0}, 1_{t>0})$ is a one-hot vector encoding the class.

Proof:

$$\begin{aligned} H(q,p) &= -\sum_{i=1}^{2} q_i \log p_i \\ &= -q_1 \log p_1 - q_2 \log p_2 \\ &= -1_{t<0} \log \frac{e^{-y}}{1+e^{-y}} - 1_{t>0} \log \frac{e^{y}}{1+e^{y}} \\ &= -\log \frac{e^{yt}}{1+e^{yt}} \\ &= -\log \frac{1}{1+e^{-yt}} \\ &= \log(1+e^{-yt}) \end{aligned}$$

## Classification Losses

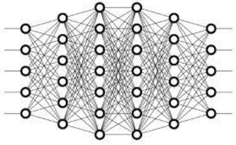
**Systematic comparison**

| | optimizable | mislabeling-robust | builds margin |
|---|---|---|---|
| 0/1 loss | ✗ | ✓ | ✗ |
| perceptron loss $\max(0, -yt)$ | ✓ | ✓ | ✗ |
| log loss $\log(1 + \exp(-yt))$ | ✓ | ✓ | ✓ |

## Handling Multiple Classes

**Blueprint:**

- Build a neural network with as many outputs as there are classes, call them $y_1, \dots, y_C$.
- Classify as $k = \arg\max[y_1, \dots, y_C]$.



**Observation:**

- The $0/1$ loss function can then be straightforwardly generalized to the multi-class case as:

| | $t=1$ | $t=2$ | $\dots$ | $t=C$ |
|---|---|---|---|---|
| $k=1$ | 0 | 1 | $\dots$ | 1 |
| $k=2$ | 1 | 0 | | |
| $\vdots$ | $\vdots$ | | $\ddots$ | |
| $k=C$ | 1 | | | 0 |

- However, this generalization of the $0/1$ loss suffers from the same problem as the original $0/1$ loss, that is, the difficulty to optimize it, and the fact it does not promote margins between the data/predictions and the decision boundary.

## Handling Multiple Classes

**Generalizing the log-loss to multiple classes:**

- Let $y_1, \dots, y_C$ be the $C$ outputs of our network. Mapping these scores to a probability vector via the softmax function

$$p_i = \frac{\exp(y_i)}{\sum_{j=1}^{C} \exp(y_j)}$$

and constructing a one-hot encoding $q$ of the class label $t$, we define the loss function as the cross-entropy $H(q,p)$, i.e.

$$\begin{aligned} \ell(y,t) = H(q,p) &= -\sum_{i=1}^{C} q_i \log p_i \\ &= -\log p_t \\ &= \log \sum_{j=1}^{C} \exp y_j - y_t \end{aligned}$$

which can be interpreted as the difference between the evidence found by the neural network for all classes, and the evidence found by the neural network for the target class.

Part 3

## Practical Aspects

## Practical Aspect 1: Non-Uniform Misclassification Costs

**Example:** medical diagnosis.

- Assumes a type of error is much more costly than another, e.g. missing the detection of a disease.

Actual

| Predicted | No infection | Infection |
|---|---|---|
| No infection | 0 | 10000 |
| Infection | 2000 | 0 |

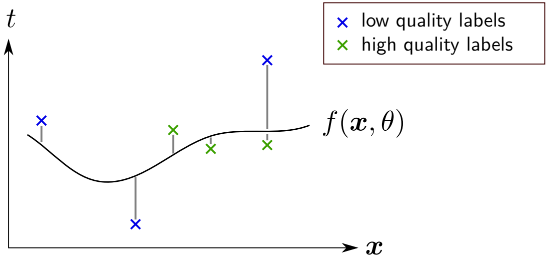
**Approach for the $0/1$ loss:**

- To reflect this cost structure, the 0/1 loss can be straightforwardly enhanced by replacing the “1”s in the loss function by the actual costs.
- Minimizing the loss function is then equivalent to minimizing the expected cost (or maximizing utility).

**Approach for other losses:**

- When the loss has a probabilistic interpretation (e.g. log-loss), one can treat predicted probabilities $p(y=k)$ as ‘ground-truth’ and estimate the expected cost for class $i$ as $\sum_{k=1}^{C} \text{cost}(\text{choose } i|k) p(y=k)$.

## Practical Aspect 2: Labels of Varying Quality



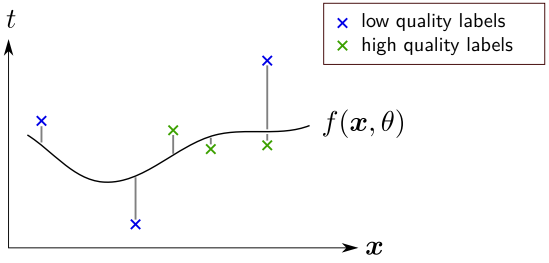
**Examples:**

- Non-expert vs. expert labeler, outcome of a physics simulation with/without approximations, noisy/clean measurement of an experimental outcome.

**Idea:**

- In presence of two similar instances that are similar but with diverging labels, focus on the high-quality one. Low-quality labels remain useful in regions with scarce data.

## Practical Aspect 2: Labels of Varying Quality



**Idea (cont.):**

- Use a different loss function for different data points, e.g. one associates to instance $i$ the loss function:

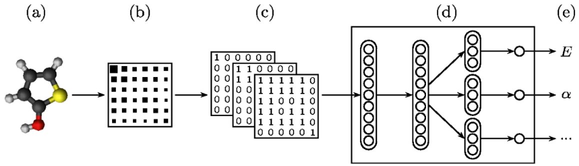
$$\ell_i(y,t) = C_i \ell(y,t)$$

where $C_i$ is a multiplicative factor set large if $i$ is a high quality data point or low if $i$ is low-quality.

## Practical Aspect 3: Multiple Tasks

In practice, we may want the same neural network to perform several tasks simultaneously, e.g. multiple binary classification tasks, or some additional regression tasks.

**Example:** (New J. Phys. **15** 095003, 2013)



Denoting by $\boldsymbol{t} = (t_1, \ldots, t_L)$ the vector of targets for the $L$ different tasks, and building a neural network with the corresponding number of outputs $\boldsymbol{y} = (y_1, \ldots, y_L)$, we can define the loss function

$$\ell(\boldsymbol{y}, \boldsymbol{t}) = \sum_{j=1}^{L} \ell_j(y_j, t_j)$$

where $\ell_j$ is the loss function chosen for solving task $j$.

## Practical Aspect 3: Multiple Tasks

**Remark 1:**

- When the different tasks are regression tasks (with similar scale and weighting), and when applying the square loss and absolute loss to these different tasks, the multi-task loss takes the respective forms:

$$\mathcal{E}(\boldsymbol{y},\boldsymbol{t}) = \sum_{l=1}^{L} (y_l - t_l)^2 = \|\boldsymbol{y}-\boldsymbol{t}\|^2$$

$$\mathcal{E}(\boldsymbol{y},\boldsymbol{t}) = \sum_{l=1}^{L} |y_l - t_l| = \|\boldsymbol{y}-\boldsymbol{t}\|_1.$$

**Remark 2:**

- We distinguish between *multi-class classification* and *multiple binary classification* tasks. For example, in image recognition, there are typically multiple objects on one image, and one often prefers to indicate for each object its presence or absence rather than to associate to the image a single class.

## Summary

## Summary

- Lectures 5–6 have highlighted that the actual data on which we train the model plays an important role. In Lecture 7, we have demonstrated that an equally important role is played by the way we specify the errors of the model through particular choices of a *loss function* $\ell$.
- Many loss functions exist for tasks such as regression, binary classification, multi-class classification, multi-task learning, etc.
- Loss functions must be designed by taking multiple aspects into account, such as the ability to account for mislabelings, the ability to tolerate some noise, and the ability to support efficient optimization.
- Loss functions can be defined flexibly to address practical aspects such as the presence of asymetric misclassification costs, subsets of the data with different data quality, or the presence of multiple subtasks.