Technische Hochschule Rosenheim

Deep Learning

Common Practices

Technische Hochschule Rosenheim
Sommer 2023
Prof. Dr. Jochen Schmidt

Acknowledgements

Many of the slides presented here are based on the Deep Learning Slides Summer Semester 2020, courtesy of
A. Maier, V. Christlein, K. Breininger, F. Denzinger, F. Thamm,
Pattern Recognition Lab, Friedrich-Alexander-University Erlangen-Nürnberg.
https://lme.tf.fau.de/

Overview

Common practices on how to
choose an architecture, **train** and **evaluate** a deep neural network:

- Training, Optimization, and Learning Rate
- Architecture Selection and Hyperparameter Optimization
- Class Imbalance
- Evaluation

Training, Optimization, and Learning Rate

Test Data

© Jonathunder, WinonaSavingsBankVault, CC BY-SA 3.0

- Overfitting is extremely easy with neural networks.
- True test set error/generalization error can be underestimated **substantially** when using the test set for model selection.
  - Attention: Choosing the architecture is the first element in model selection
    → should never be done on the test set.
- Do initial experimentation on a small subset of the dataset!

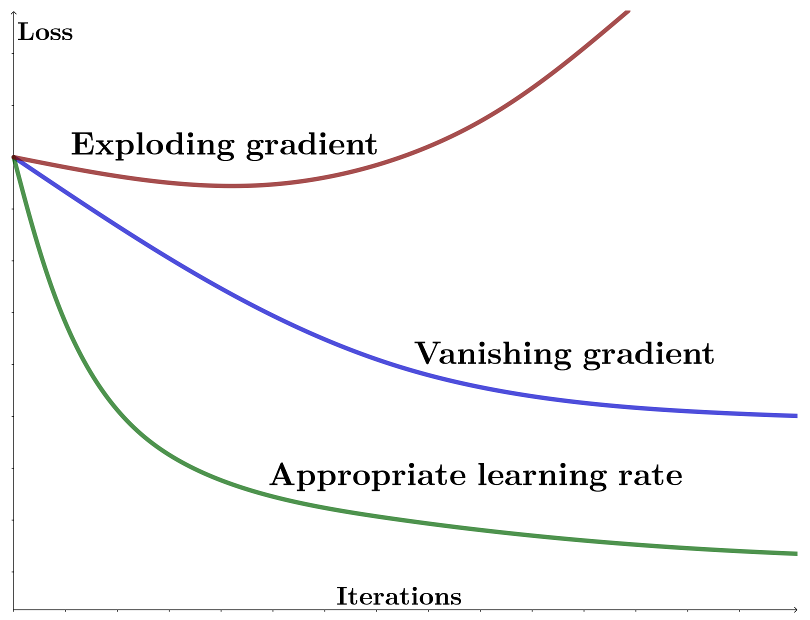
"Ideally, the test set should be kept in a vault, and be brought out only at the end of the data analysis."
T. Hastie, R. Tibshirani, J. Friedman: The Elements of Statistical Learning

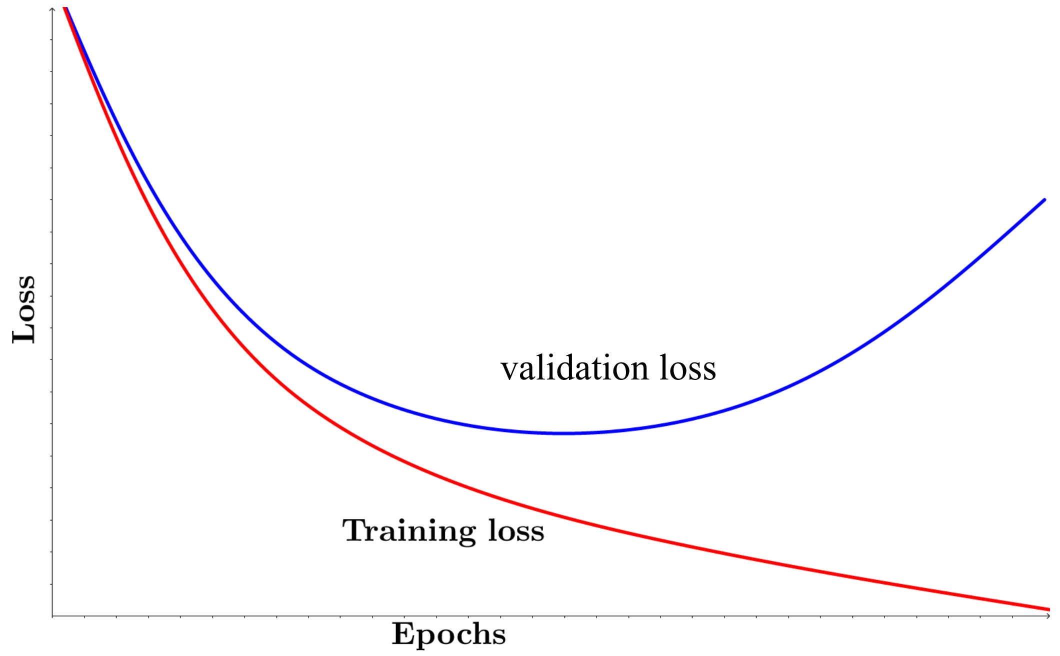
Before Training: Training!

Goal: Check whether the architecture is **in general** capable to learn the task.

- Before training the network on the full training data set, take a small subset (5-20 samples) and try to **overfit** the network to get zero loss.
  - Optionally: Turn off regularization that may hinder overfitting.
- If the network cannot overfit:
  - Bug in the implementation.
  - Model too small → increase number of parameters.
  - Model not suitable for the task.
- Also: Get a first idea about how the data, loss and network behave.

During Training: Monitor Loss Function



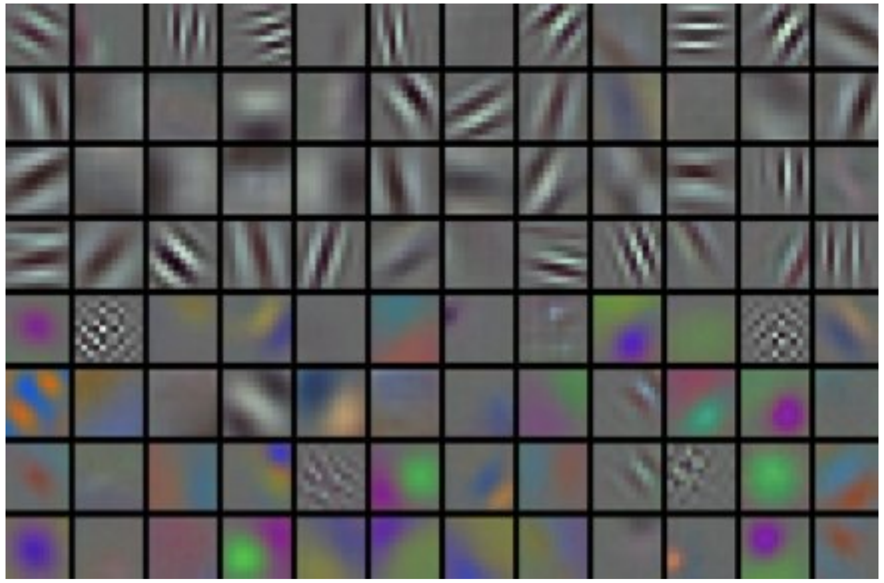


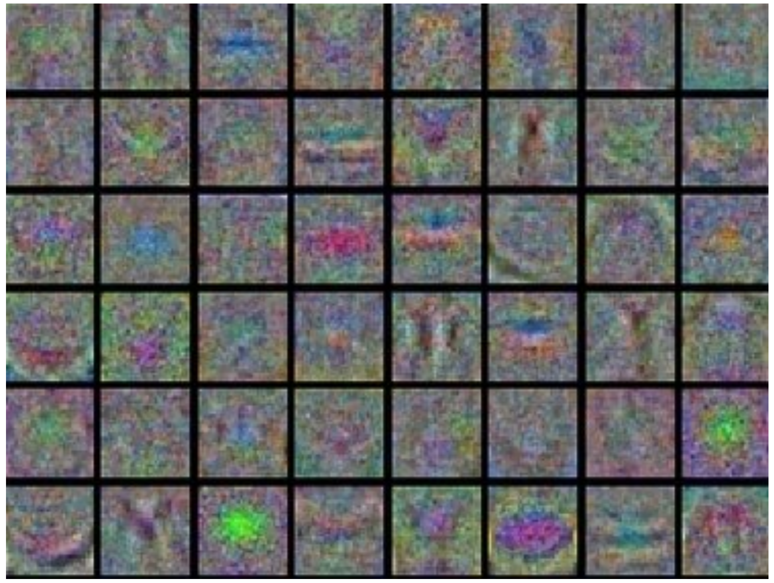
- Check learning rate (→ upcoming),
- Identify large jumps in the learning curve,
- Very noisy curves → increase batch size.

- Monitor amount of overfitting of the network.
- If training and validation loss diverge: overfitting → increase regularization/ early stopping
- If training and validation loss are close but high: underfitting → decrease regularization/ increase model size
- Save intermediate models if you want to use them for testing!

During Training: Monitor Weights and Activations

- Track relative magnitude of the weight update: Should be in a sensible range (approx. 10^{-3}).
- Check for very large or saturated activations ($\rightarrow$ dying ReLUs)
- Convolutional layers: check filters of the first few layers. Should develop towards smooth and regular filters.





Source: http://cs231n.github.io/neural-networks-3/

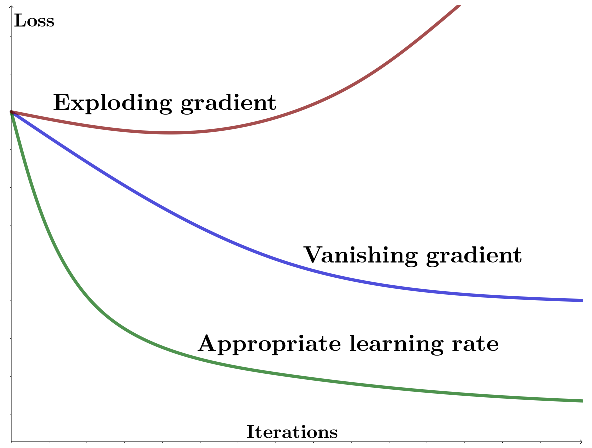
Choosing an Optimizer

- Batch gradient descent: Requires large memory, too slow, too few updates.
- Stochastic gradient descent (SGD): loss function and gradient become very noisy if only one/few samples are used.
- SGD with mini-batches: “best of both worlds”
  - Frequent, more stable updates.
  - Gradient noisy enough to escape local minima.
  - Adapting mini-batch size yields smoother/noisier gradient.
  - Addition of momentum prevents oscillations and speeds up optimization.

- Recommendation: Start with Mini-Batch SGD + momentum.
- For faster convergence speed → ADAM.

Learning Rate: Observing the Loss Curve

- Learning α rate has a large impact on the successful training of a network.
- For almost all gradient based optimizers, α has to be set manually.
- Effect of learning rate is often directly observable in the loss curve.



- But this is a very simplified view!
- We want an adaptive learning rate: Progressively smaller steps to find the optimum
  $\rightarrow$ Annealing the learning rate

Annealing the Learning Rate

- In deep learning context often known as **learning rate decay**.
  - Decay means yet another hyper-parameter.
  - We need to avoid oscillation as well as a too fast cool down!
- Decay strategies:
  - **Stepwise decay**: Every n epochs, reduce learning rate by a certain factor, e.g., 0.5, or by a constant value, e.g., 0.01.
    Variant: Reduce learning rate when **validation error** stagnates.
  - **Exponential decay**: At epoch t: $\alpha \coloneqq \alpha_0 e^{-kt}$ with k controlling the decay.
  - **$^1/_t$ -decay**: At epoch t: $\alpha \coloneqq {}^{\alpha_0}/_{(1+kt)}$.
- Stepwise decay is most common: hyperparameters are easy to interpret.

- Second-order methods are currently uncommon in practice
  - computationally very expensive
  - and therefore they do not scale well.

Architecture Selection and Hyperparameter Optimization

Reminder

© Jonathunder, WinonaSavingsBankVault, CC BY-SA 3.0

Test data → vault

Hyperparameter Optimization

Neural networks have an enormous amount of hyperparameters

- Architecture:
  - Number of layers & number of nodes per layer
  - Activation function
  - …
- Optimization
  - Initialization
  - Loss function
  - Optimizer (SGD, Momentum, ADAM, …)
  - Learning rate, decay, batchsize
  - …
- Regularization
  - Regularizer, e.g., L_2 -, L_1 -loss
  - Batch normalization?
  - Dropout?
  - …
- …

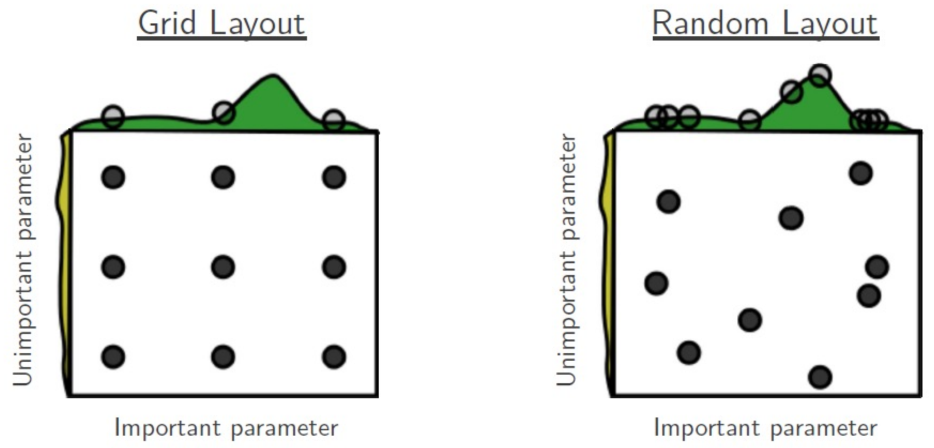
Choosing Architecture and Loss Function

- First step: Think about the problem and the data:
  - What could the features look like?
  - What kind of spatial correlation do you expect?
  - What data augmentation makes sense?
  - How will the classes be distributed?
  - What is important regarding the target application?

- Start with simple architectures and loss functions.
- Do your research: Try **well-known** models first and foremost!
- If you change/adapt the architecture: Find reasons why the network should perform better.

Hyperparameter Search

- Learning rate, decay, regularization/dropout etc. can be tuned more easily.
- Still, networks can take days/weeks to train.
- Search for hyperparameters using a log scale (e.g., $\alpha \in \{0.1, 0.01, 0.001\}$).
- Options: **Grid search** or **random search**:
  - Use random search instead of grid search [Ber12]:
    - Easier to implement.
    - Better exploration of parameters that have strong influence on the result.



Hyperparameter Search – Coarse to Fine Search

- Hyperparameters are highly interdependent.
- Optimize on a coarse to fine scale:
  - Training network only for a few epochs.
  - Bring all hyperparameters in sensible ranges.
  - Then refine using random/grid-search.

Class Imbalance

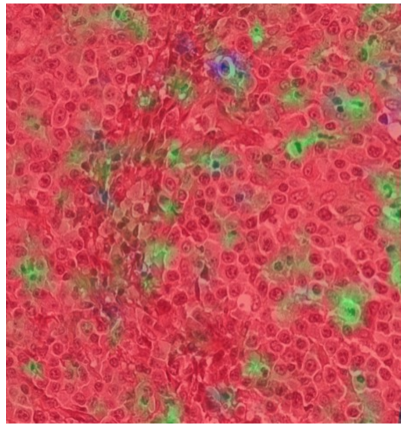
Motivation

- Often, different classes occur with very different frequencies in the data set.
  - This is a big challenge for machine learning algorithms.

© Lotus Head, Credit-cards, CC BY-SA 3.0

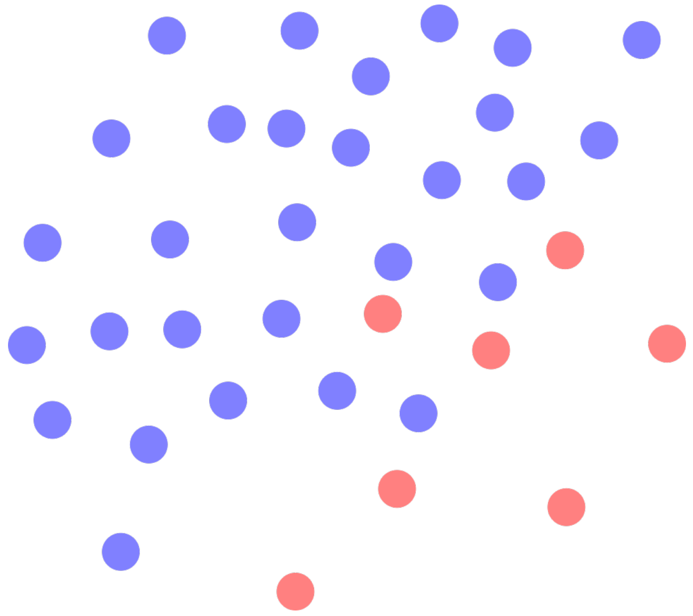
- Example 1: Fraud detection
  - Out of 10,000 transactions, 9,999 are genuine and 1 is fraudulent:
  - Classifying every transaction as genuine: 99.99% accuracy
  - Or, less extreme, using a method that misclassifies 1 out of 100 genuine transactions: 99% accuracy

- Example 2: Detect mitotic cells for tumor diagnostics [Aub17]
  - Problem: Mitotic cells only make up a very small portion of cells in tissues.
  - Data of a certain class is seen much less during training.



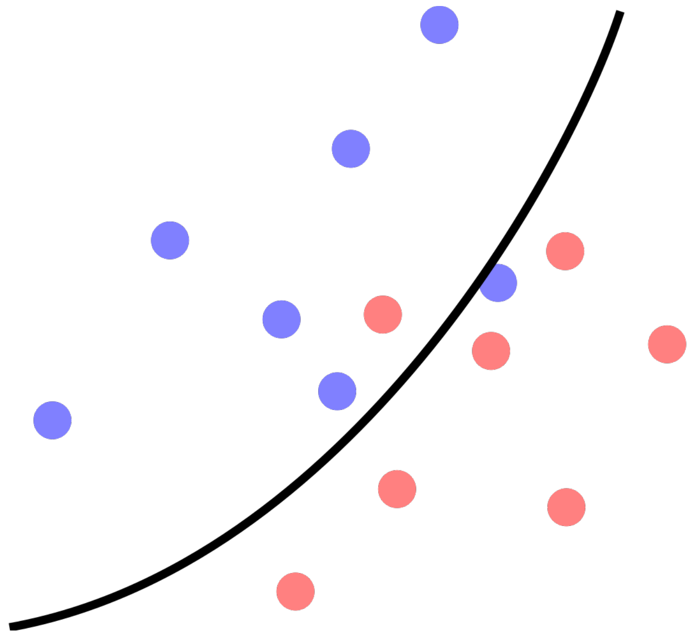
Resampling Strategies for Class Imbalance

Idea: Balance class frequencies by sampling classes differently.



Resampling Strategies – Undersampling

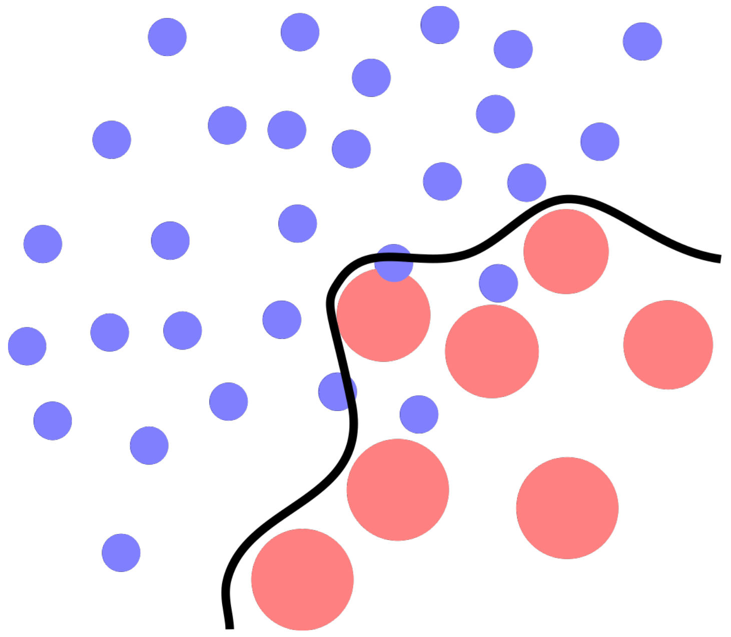
- In each iteration, take a subset of the overrepresented class.
- Samples of all classes are now presented to the network equally often.
- Disadvantage: Not all available data is used for training and can lead to underfitting.



Resampling Strategies – Oversampling

- Use sample from underrepresented class multiple times.
- All available data can be used.
- Disadvantage: Can lead to overfitting.

- Also possible: Combine Under- and Oversampling.



Resampling Strategies for Class Imbalance

- More advanced resampling strategies available that try to avoid the shortcomings of simple under-/oversampling, e.g., Synthetic Minority Over-Sampling Technique (SMOTE).
    - Rather uncommon in deep learning.
- Underfitting caused by undersampling can be reduced by taking a different subset after each epoch.
- Data augmentation can help to reduce overfitting for underrepresented class.

Class Imbalance – Adapt the Loss Function

- Instead of “fixing” the data, adapt the loss function to be stable with respect to class imbalance.
- Weigh loss with inverse class frequency w_k, e.g., weighted cross entropy:

$$-w_k y_k \ln \hat{y}_k$$

- Instead of class frequency, weights can be adapted with regards to other considerations.

Technische Hochschule Rosenheim

Evaluation

Performance Evaluation

- Network was trained on training set, hyper-parameters estimated on the validation set.

- Evaluate generalization performance on previously unseen data: the test set.
- We can now open the vault!

VAULT DOOR
WEIGHT: 22 1/2 Tons
THICKNESS: 22 Inches
STEEL: 11 Layers of Special Cutting and Drill Resistant
LOCKS: 4 Hamilton Watch Movements for Time Locks

© Jonathunder, WinonaSavingsBankVault, CC BY-SA 3.0

Of All Things the Measure is Man*

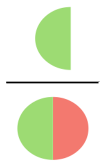
- Data is annotated and labeled by humans.
- During training, all labels are assumed to be correct ⚡ "to err is human"
- Additionally: Ambiguous data.
- Multiple human voters: Take mean (if possible) or majority vote.
- Steidl et al. [Ste05]: Entropy-based measure that takes "confusions" of human reference labelers into account:
  - Humans confuse certain classes more often than others (Angry vs. Happy/Angry vs. Annoyed)
  - Mistakes by the classifier are less severe if the same classes are confused by humans.

*Protagoras of Abdera (c.490 - c.411 BC)

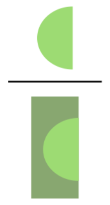
Performance Measures – Binary Classification

- We have
  - Total number of Positives/Negatives: P/N
  - True/False Positives: TP/FP
  - True/False Negatives: TN/FN

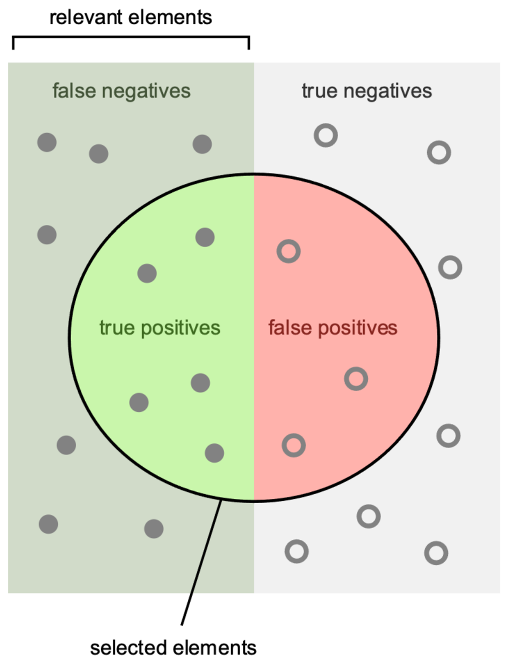
- Accuracy: $ACC = \frac{TP+TN}{P+N}$
- Precision/positive predictive value: $precision = \frac{TP}{TP+FP}$ =



- Recall/true positive value: $recall = \frac{TP}{TP+FN}$ =



- Specificity/true negative value: $specificity = \frac{TN}{TN+FP}$
- F1-score: $F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$
- Receiver operating characteristic (ROC) curve
  - Area Under Curve (AUC)



© Walber, Precisionrecall, CC BY-SA 4.0

Performance Measures – Multiple Classes

- Adapted versions of measures mentioned above.
- Top-K error: True class label is not in the K classes with the highest prediction score.
  - Common: Top-1 and Top-5 error.
  - Example: ImageNet performance usually measured with Top-5 error.
- Confusion matrix

Cross Validation

- k-fold cross validation:
  - Split data in k folds.
  - Use k−1 folds as training data, test on fold k.
  - Repeat k times.
- Rather uncommon in deep learning due to long training times.
- Can be used for hyperparameter estimation (nested!), or to evaluate stability of (hyper-)parameters.
  - Attention: almost always additional bias (when using it for architecture selection, hyperparameters).
  - Underestimates variance of results: Training runs are not independent.
- Even without cross-validation: Training is a highly stochastic process.

→ Retrain network multiple times and report average performance and standard deviation.

Comparing Classifiers

Example: Is my new method with 91.5% accuracy better than the state-of-the-art with 90.9%?

- Training a neural network is a stochastic process.
- Simply comparing two (or more) numbers yields biased results!

Actual question: Is there a **significant** difference between classifiers?

- Run training for each method/network multiple times.
- Determine whether performance is significantly different, e.g., **Student's t-test**!
  - Compares two normally distributed data sets with equal variance.
  - Determines whether the means are significantly different with respect to a **significance level** α (e.g., 5% or 1%).

Comparing Classifiers – Bonferroni Correction

- Significance level means: The probability that this difference is caused by **chance** $< \alpha$.
- If we compare several classifiers trained **on the same data**, this chance can rise significantly!
- Correct for multiple tests using Bonferroni correction:
  - For n tests with significance level α the total risk is $n\alpha$.
  - To reach a total significance level of α, we have to choose an adjusted $\alpha' = {}^{\alpha}/_{n}$ for each individual test.
- Assumes independence between tests: Pessimistic estimation of significance.
- More accurate, but incredibly time-consuming: Permutation tests [Dic11]

Summary

- Check your implementation before training: Gradient, initialization, …
- Monitor training process continuously: training/validation loss, weights, activations.
- Stick to established architectures before reinventing the wheel.
- Experiment with few data sets, keep your test data safe until evaluation.
- Decay the learning rate over time.
- Do random search (not grid search) for hyperparameters.
- Check for significance when comparing classifiers.

References

[Aub17]M. Aubreville, M. Krappmann, C. Bertram, et al. “A Guided Spatial Transformer Network for Histology Cell Differentiation”. In: ArXiv e-prints (July 2017). arXiv: 1707.08525.

[Ber12] James Bergstra and Yoshua Bengio. “Random Search for Hyper-parameter Optimization”. In: J. Mach. Learn. Res. 13 (Feb. 2012), pp. 281–305.

[Dic11] Jean Dickinson Gibbons and Subhabrata Chakraborti. “Nonparametric statistical inference”. In: International encyclopedia of statistical science. Springer, 2011, pp. 977–979.

[Ste05] Stefan Steidl, Michael Levit, Anton Batliner, et al. “Of All Things the Measure is Man: Automatic Classification of Emotions and Inter-labeler Consistency”. In: Proc. of ICASSP. IEEE, Mar. 2005.