Common Practices

A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul, M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
May 28, 2020

Outline

Recap

Training Strategies

Optimization and Learning Rate

Architecture Selection and Hyperparameter Optimization

Ensembling

Class Imbalance

Evaluation

Recap

Training a Neural Network

- So far: all the nuts and bolts about how to train a network:
  - Fully connected and convolutional layers
  - Activation function
  - Loss function
  - Optimization
  - Regularization
- Today: Common practices on how to **choose an architecture**, **train** and **evaluate** a deep neural network.

First Things First: Test Data

"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

First Things First: Test Data (cont.)

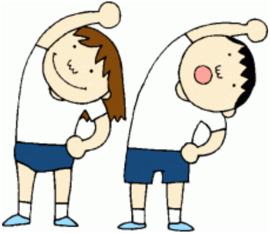
- Overfitting is extremely easy with neural networks (see e.g. ImageNet with random labels [5]).
- True test set error/generalization error can be underestimated **substantially** when using the test set for model selection!
- Attention: Choosing the architecture is the first element in model selection ➔ should never be done on the test set!
- Do initial experimentation on smaller subset of the dataset!

Training Strategies

Before Training: Gradient Checks

Own loss function, own layer implementation etc.: Check correct computation of gradient by comparing analytic and numerical gradient.

- Use centered differences for numeric gradient.
- Use relative error instead of absolute differences.
- Numerics:
  - Use double precision for checking.
  - Temporarily scale loss function if you observe very small values ($< 1e-9$).
  - Choose h appropriately.



Source: https://fhsperformance.wordpress.com/2016/09/02/ramp-up-your-warm-ups/

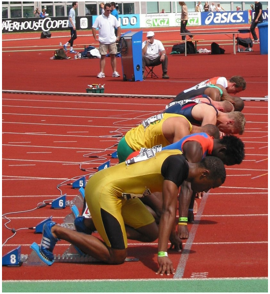
Before Training: Gradient Checks (cont.)

Additional recommendations:

- Use only a few datapoints ➔less issues with non-differentiable parts of the loss function.
- Train the network for a short period of time before performing gradient checks.
- Check gradient first without, then with regularization terms.
- Turn off data augmentation and dropout.

Before Training: Check Initalization and Loss

- Goal: Check correct random initialization of layers.
- Compute the loss for each class on the **untrained network**, with regularization turned off.
- Compare loss with loss achieved when deciding for a class randomly (**chance**).
- Repeat with multiple random initializations.



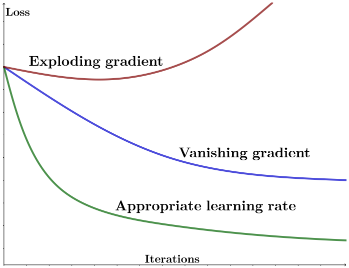
Source: https://commons.wikimedia.org/

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.

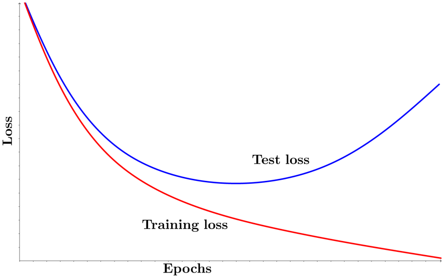
During Training: Monitor loss function

- Recap:



- Check learning rate (➔more in a bit).
- Identify large jumps in the learning curve.
- Very noisy curves ➔increase batch size.

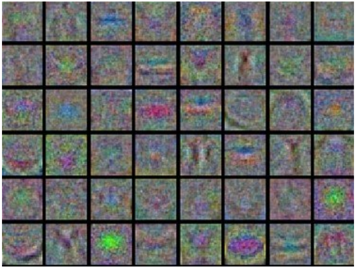
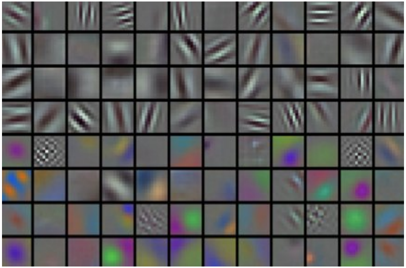
During Training: Monitor Validation Loss



- 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. 1e-3).
- Convolutional layers: check filters of the first few layers. Should develop towards smooth and regular filters.
- Check for very large or saturated activations (➔dying ReLUs)

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

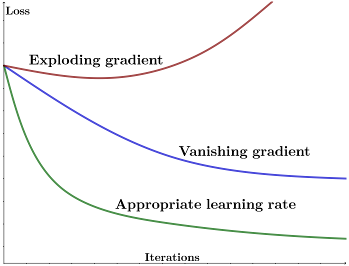
Optimization and Learning Rate

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/more noisy gradient.
- Addition of momentum prevents oscillations and speeds up optimization.
- Effect of hyper-parameters relatively straight forward.
- Recommendation: Start with Mini-Batch SGD + momentum.
- For faster convergence speed →ADAM.

Learning rate: Observing the loss curve

- Learning rate η has a large impact on the successful training of a network.
- For almost all gradient based optimizers, η has to be set.
- 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

→ **Annealing** the learning rate.

Annealing the Learning Rate

- In deep learning context often known as **learning rate decay**.
- Decay means yet another hyper-parameter.
- Need 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: $\eta = \eta_0 e^{-kt}$ with k controlling the decay.
  - **$1/t$-decay**: At epoch t: $\eta = \eta_0 / (1 + kt)$.
- Stepwise decay most common: hyper-parameters are easy to interpret.
- Second-order methods are currently uncommon in practice, as they do not scale as well.

NEXT TIME

ON DEEP LEARNING

Common Practices - Part 2

A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul, M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
May 28, 2020

FRIEDRICH-ALEXANDER-UNIVERSITÄT ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Architecture Selection and Hyperparameter Optimization

Reminder

Test data →vault!

Hyperparameter optimization

Neural networks have an enormous amount of hyperparameters.

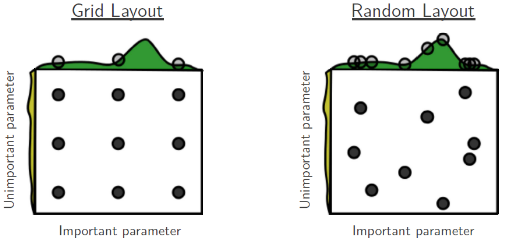
- Architecture:
  - Number of layers & number of nodes per layer
  - Activation function
  - ...
- Optimization
  - Initialization
  - Loss function
  - Optimizer (SGD, Momentum, ADAM, ...)
  - Learning rate, decay & batch size
  - ...
- Regularization
  - Regularizer, e.g., L_2-, L_1-loss
  - Batch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

Choosing Architecture and Loss Function

- First step: Think about the problem and the data:
  - How 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., $\eta \in \{0.1, 0.01, 0.001\}$).
- Options: Grid search or random search:
  - Use random search instead of grid search [2]:
  - ➔ Easier to implement.
  - ➔ Better exploration of parameters that have strong influence on the result.



Source: [2]

Hyperparameter search: Coarse to fine search

- Hyperparameters 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.

Ensembling

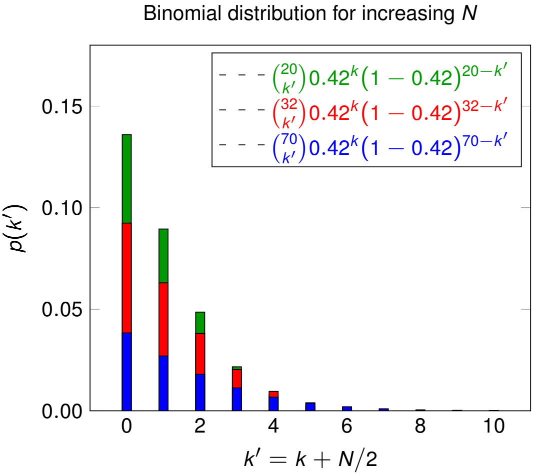
Concept

- So far we have always considered a **single** classifier. Can't we get better by using **many**?
- Assume N classifiers **independently** performing a correct prediction with probability $1 - p$
- The probability of seeing k errors is:

$$\binom{N}{k} p^k (1-p)^{N-k},$$

  known as binomial distribution
- So the probability of a majority $k > \frac{N}{2}$ to be wrong is:

$$\sum_{k>\frac{N}{2}}^{N} \binom{N}{k} p^k (1-p)^{N-k}$$



- $\sum_{k>\frac{N}{2}}^{N} \binom{N}{k} p^k (1-p)^{N-k}$ **monotonically decreasing** for $N \to \infty$
- **Accuracy** $\to 1$!
- The big assumption here is **independence**

Concept (cont.)

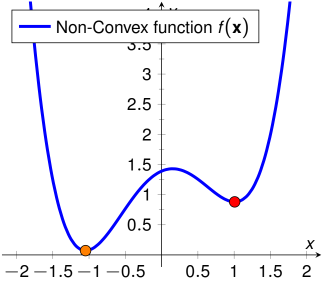
Ensembling

- Produce *N* **independent** classifiers/regressors
- **Combine** their **predictions** by majority/averaging

How to produce the components?

- Different **models**

Local Minima



- Can we use multiple local minima we get during training?
- Combine models across optimization process
- Can be combined with a **cylic** learning rate

Concept (cont.)

Ensembling

- Produce N **independent** classifiers/regressors
- **Combine** their **predictions** by majority/averaging

How to produce the components?

- Different **models**
- Different model **checkpoints**
- **Moving average** of $\mathbf{w}$ [6]
- Different **methods**

➔ **Easy performance boost if you need just a bit more**

NEXT TIME

ON DEEP LEARNING

Common Practices - Part 3

A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul, M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
May 28, 2020

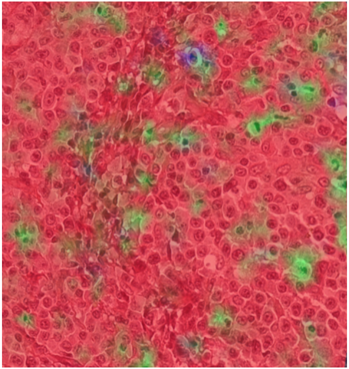
Class Imbalance

Motivation

- Often, different classes are available with very different frequencies in the data set.
- Big challenge for machine learning algorithms.
- Example 1: Fraud detection
  - Out of 10000 transactions, 9999 are genuine and 1 is fraudulent:
  - ➔ Classifying every transaction as genuine: 99.99% accuracy
  - ➔ Misclassifying 1 out of 100 genuine transactions: 99% accuracy

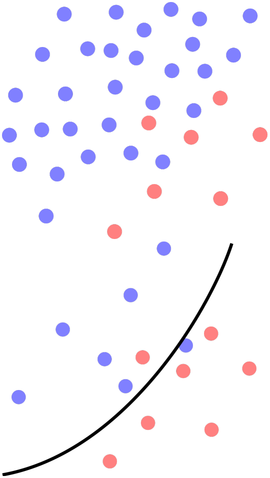
Motivation (cont.)

- Task: Detect mitotic cells for tumor diagnostics [1].
- 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.
- Measures like accuracy, L_2 norm, cross-entropy do not show imbalance.



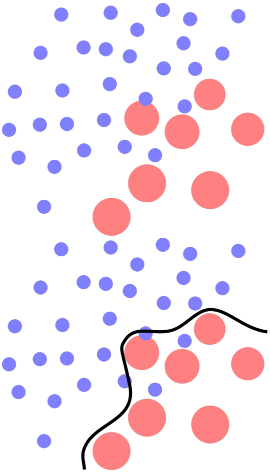
Resampling Strategies for Class Imbalance

- Idea: Balance class frequencies by sampling classes differently.
- **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 for Class Imbalance (cont.)

- **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 (cont.)

- 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 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.
- Weight loss with inverse class frequency, e.g., weighted cross entropy:

$$L(\mathbf{y}, \hat{\mathbf{y}}) = -w_k \log(\hat{y}_k)\big|_{y_k=1} \tag{1}$$

- More common in segmentation problems: Dice-loss based on Dice coefficient.
- Instead of class frequency, weights can be adapted with regards to other considerations.

NEXT TIME

ON DEEP LEARNING

Common Practices - Part 4

A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul, M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
May 28, 2020

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!

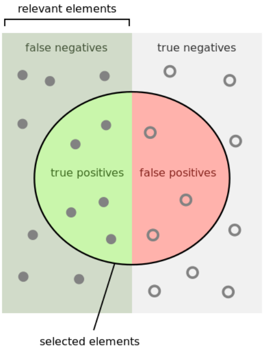
Source: de.disney.wikia.com/wiki/Dagobert_Duck

Of All Things the Measure is Man [8]

- Protagoras of Abdera (c.490 - c.420 BCE)
- Data is annotated and labeled by humans.
- During training, all labels are assumed to be correct $\lightning$ "to err is human"
- Additionally: Ambiguous data.
- Multiple human voters: Take mean (if possible) or majority vote.
- Steidl et al. 2005: Entropy-based measure that takes "confusions" of human reference labelers into account.
  - Humans confuse certain classes with each other more (Angry vs. Happy/Angry vs. Annoyed)
  - Mistakes by the classifier are less severe if the same classes are confused by humans.

Performance measures

- Classification problem →classification measures:
- Binary classification problem:
  - True/False Positives: TP/FP
  - True/False Negatives: TN/FN
- Accuracy: $\mathrm{ACC} = \frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{P}+\mathrm{N}}$
- Precision/pos. predictive value: $\mathrm{precision} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}$
- Recall/true positive value: $\mathrm{recall} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$
- Specificity/true negative value: $\mathrm{specificity} = \frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FP}}$
- F1-score: $\mathrm{F1} = 2 \cdot \frac{\mathrm{TPV} \cdot \mathrm{TNV}}{\mathrm{TPV}+\mathrm{TNV}}$
- Receiver operating characteristic (ROC) curve.



Source: https://commons.wikimedia.org/

Performance measures: Multiclass classification

- 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.

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.
- Underestimates variance of results: Training runs are not independent.
- Attention: almost always additional bias (architecture selection, hyperparameters).
- 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. 0.05 or 0.01).

Comparing Classifiers: Bonferroni Correction

- Interpretation: The probability that this difference is caused by **chance** $< \alpha$.
- If we compare multiple classifiers, this chance can rise significantly due to **multiple comparisons**!
- Correct for multiple tests using Bonferroni correction:
  - For *n* tests with significance level α, the total risk is $n \cdot \alpha$.
  - → To reach a total significance level of α, choose adjusted $\alpha' = \alpha/n$ for each individual test.
- Assumes independence between tests: Pessimistic estimation of significance.
- More accurate, but incredibly time-consuming: Permutation tests [3].

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 evaluation data safe until evaluation.
- Decay the learning rate over time.
- Do random search (not grid search) for hyperparameters.
- Perform model ensembling for better performance.
- Check for significance when comparing classifiers.

NEXT TIME

ON DEEP LEARNING

Coming Up

Evolution of neural network architectures:

- From deep networks to deeper networks.
- From “sparse” to dense connections.
- LeNet, GoogLeNet, ResNet, . . .

Further Reading

- Link SGD Tricks by Leon Bottou.
- Link: Interesting loss functions.
- Link: Practical recommendations by Yoshua Bengio (from 2012).

References

References I

[1] 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 [cs.CV].

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

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

[4] Yoshua Bengio. "Practical recommendations for gradient-based training of deep architectures". In: Neural networks: Tricks of the trade. Springer, 2012, pp. 437–478.

References II

[5] Chiyuan Zhang, Samy Bengio, Moritz Hardt, et al. "Understanding deep learning requires rethinking generalization". In: arXiv preprint arXiv:1611.03530 (2016).

[6] Boris T Polyak and Anatoli B Juditsky. "Acceleration of stochastic approximation by averaging". In: SIAM Journal on Control and Optimization 30.4 (1992), pp. 838–855.

[7] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. "Searching for Activation Functions". In: CoRR abs/1710.05941 (2017). arXiv: 1710.05941.

[8] 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 â€“ Institute of Electrical and Electronics Engineers, Mar. 2005.