MA Presentation

1. **Introduction slide**
2. **Hyperparameter Optimization**

   - Definition Hyperparameters: Parameters of a ML model that are fixed before training, number of layers/ neurons, epochs, learning rate, … → architecture decision (size of network etc.)
   - Optimization: finding the optimum of a function, e.g. Rosenbrock function → optimal value at (1,1)

3. **Hyperparameter Optimization**

   - Together: Optimization of a very expensive black box function
   - Each function evaluation is training & evaluation of a ML model with different configuration
   - Example: 2D optimization, 2-layer neural network, diamonds dataset used → mean average percentage error on test set
   - Problem: Too expensive to evaluate this many times to get this plot
   - Already many approaches, baby sitting (trial & error like everyone does it), expert knowledge, more epochs → lower learning rate etc. …)

4. **Hyperparameter Optimization – State of the Art**

   - Techniques from literature, without any knowledge about hyperparameters & model
   - Grid search: spanning a homogeneous grid on the whole domain
     - curse of dimensionality
     - only 7 different values for each parameter
   - Random search: pick random candidates (uniformly on whole domain)
     - more different values for each hyperparameter

5. **Hyperparameter Optimization – State of the Art**

   - Bayesian Optimization: iterative algorithm
   - three parts:
     - archive: all function evaluations so far
     - surrogate model: approximates the function with regression, e.g. gaussian process, random forest,…
     - acquisition function: its maximum is the new candidate point → added to the archive

6. **Hyperparamater Optimization with Sparse Grids**

   - Efficient way for representing functions:
     - number of grid points scales exponentially vs. not exponentially
     - but on the other hand, sparse grids not as precise as full grids
   - interpolation with basis functions, B-splines very good for optimization
   - Sparse Grid search: new approach, iterative, adaptive to the function values

7. Sparse Grid Hyperparameter Optimization

- In each iteration, one point is selected to be refined with the Novak-Ritter refinement criterion:
- For each point, term is evaluated and point with minimal value is selected:
  - **r**: rank of the points → place of the function value in the ascending order of all function values
  - **l:** level of the point
  - **d:** degree, number of previous refinements of this point
- Adaptivity parameter: Trade-off between exploration (high value) and exploitation (low value)

8. Adaptivity Parameter

- Smaller value (top): very adaptive sparse grid, small value found in the top right corner, few points in other regions
- higher value (bottom): a bit adaptive, most points in the top regions, but more distributed
- Extreme case: gamma = 1 → normal sparse grid without any adaptivity
- Perfect parameter depends on setting (dataset & model), but 0.85 is a good value to start with
- Interpolation possible with B-splines basis functions

9. Sparse Grid Hyperparameter Optimization

- Not only possible to take the best grid point as result, but one step further
- Use interpolant and optimize
- Gradient-free optimizers, not using the gradient of the interpolant
- Gradient-based optimizers: Gradient descent (as depicted)
- Additionally global, globalized, and local optimizers
- Gradient descent steps with different number of grid points → have to use high number of grid points to really optimizer
- Problem in our setting: too few grid points because evaluation very expensive
- But: additionally apply global and local optimizer to the interpolant and take smallest of all three values as the result

10. Numerical Results (2D)

- 2 Hyperparameters: Epochs and learning rate
- 3 different datasets: house_16H, house_sales, and Brazilian_houses
- Mean average percentage error
- Trends: decrease with increasing cost (budget given)
- Sparse grids optimization very competitive, best result found for second and third dataset and cost higher than 40

11. Numerical Results (3D)

- Additionally optimize the batch size
- Higher dimension: problems for grid search only 3, and 4 values per parameter → 27 and 64 cost
- For house_sales dataset: sparse grid optimization very powerful already for small budget
- for house_16H, it need a few iterations to find a good optimum

12. Numerical Results (5D)

- Additionally optimize the number of layers and the neurons per layer → architecture decisions
- big problems for the grid search, only 2 values for each parameter → 2^5=32 budget
- sparse grid search performs well for all three datasets

13. Application: MNIST

- Handwritten digits classification
- Also different type of network: convolutional
- 9 dimensional problem, architecture decision and parameter values
- Grid search only performs with budget of 1
- Others: Reach accuracy of up to 97.5% with cost of 80-93%
- Bayesian optimization very fast very good
- Sparse grid search a bit faster than random search
- Note: Still only a small and simple network, not a specific network architecture for object classification, and limited number of epochs

14. Iterative Adaptive Random Search

- Second new approach for hyperparameter optimization
- Idea: Combine advantages of random search and iterative optimization algorithm
- Algorithm:
  - Start: Sample initial points uniformly in whole domain
  - Iterations until the budget is exhausted:
    - Sort the array of all sampled points with increasing function values
    - Select point to be refined → **refinement criterion**
    - Sample new points according to **refinement strategy**
  - Return sorted array
- Two questions: **Refinement criterion** and **refinement strategy**
- Important parameters:
  - Number of initial points
  - Number of points added per refinement
  - adaptivity parameter
  - refinement strategy
  - budget

15. Refinement Strategies

- Each case: 5 initial points were sampled
- Refinement criterion is used, similar to Ritter-Novak refinement criterion, the point with minimal term
- Interval based refinement: In each dimension, the upper and lower bounds are defined by the next neighboring point, then sampled uniformly in blue area
- Uniform d-ball sampling: radius calculated with distances to all points, not exactly the next point
- Normal distribution sampling: similar to d-ball sampling, but with probabilities, idea: if the point is promising, we should add points next to it with higher probability

16. Adaptivity Parameter gamma = 1

- Left part is constant → not adaptive point generation
- points distributed over domain, but: depends on initial distribution, e.g. areas without any points

17. Adaptivity Parameter gamma = 0

- Right part is constant → point wih smallest rank is always reifned
- Very adaptive
- Difference between 2 and 3:
  - 2 has concentrated ball with sharp border
  - 3 has more points also outside → can see the normal distribution

18. Comparison of Optimization Algorithms 2D – only Random Searches

- 2D optimization problem, epochs and learning rate
- Focus on Random search and strategies:
  - In the beginning conventional is best
  - Then: Strategy 3 with smallest error

19. Comparison of Optimization Algorithms 2D

- All in all best: sparse grid opt after 60 best solution

20. Comparison of Optimization Algorithms 6D

- 6D optimization problem, additionally batch size, number of layers, neurons per layer, dropout probability
- Random search outperforms all refinement strategies
- Reason: d-ball sampling (and normal distribution sampling)

21. Discussion – Sparse Grid Hyperparameter Optimization

- Number of grid points often not high enough for interpolated function to be good enough
- Example image: interpolated function has minimal point in right top corner
- Nevertheless: apply global and local optimizer, just compare to the best grid point
- Problems if optimum at the boundary: optimizer can sometimes help
- Or use sparse grid with boundary points, BUT: number of function evaluations increases
- ALL IN ALL: time to solution is most important and exact optimum is not that important, also because of stochastic behavior
  → initialization, splitting, …

22. Discussion – Iterative Adaptive Random Search

- Promising results in small dimension
- **Problem**: Volume of d-ball compared to d-hypercube decreases rapidly for increasing d
- → worse performance than conventional random search for higher dimensions
- More analysis:
  - Convergence?
  - Adapt algorithm?
  - Other search radius?
  - Change refinement criterion (e.g. add function value)?
  - Other refinement strategy?

23. Conclusion

24. Optimization – Categorical Hyperparameters

- Examples: choice of optimizer and also rounding to integer values
- → always discretization
- In this case: extreme case with only 3 different values for each parameter
- The more grid points, the exacter the interpolated function
- Interpolated function only important for additional optimizer → too few points to rely on optimizer anyway

25. Appendix

- Ritter-Novak criterion JULIAN VALENTIN:

  Criterion. The Novak–Ritter refinement criterion [Nov96] refines the grid point $\boldsymbol{x}_{\ell,i}$ that minimizes the product[1]

  $$(r_{\ell,i}+1)^{\gamma}\cdot(\|\boldsymbol{\ell}\|_1+d_{\ell,i}+1)^{1-\gamma}. \tag{5.7}$$

  Here, $r_{\ell,i} := |\{(\boldsymbol{\ell}',\boldsymbol{i}') \in K \mid f(\boldsymbol{x}_{\ell',i'}) \leq f(\boldsymbol{x}_{\ell,i})\}| \in \{1,\dots,|K|\}$ is the *rank* of $\boldsymbol{x}_{\ell,i}$ (where K is the current set of level-index pairs of the grid), i.e., the place of the function value at $\boldsymbol{x}_{\ell,i}$ in the ascending order of the function values at all points of the current grid. We denote the *degree* $d_{\ell,i} \in \mathbb{N}_0$ of $\boldsymbol{x}_{\ell,i}$ as the number of previous refinements of this point. Finally, $\gamma \in [0,1]$ is the *adaptivity parameter.* We have to choose a suitable compromise between exploration ($\gamma = 0$) and exploitation ($\gamma = 1$). The best choice of course depends on the objective function f at hand, but for our purposes, we choose a priori a value of $\gamma = 0.15$. However, it may be an option to adapt the value of γ automatically during the grid generation phase.

- Uniform d-ball sampling:

```
# an array of d normally distributed random variables
u = np.random.normal(0, 1, len(self.hyperparameterspace))
norm = np.sum(u**2) ** (0.5)
r = random.uniform(0, 1)**(1.0/len(self.hyperparameterspace))
x = r*u/norm
x = x*smallest_dist # scaling
```

- Probabilities for rejection sampling of d-ball with 1 radius, acceptance rate:

1D	100%
2D	78%
3D	52%
10D	0.25%

- Dropout probability not 1:
  “Inputs not set to 0 are scaled up by 1/(1 - rate) such that the sum over all inputs is unchanged.” - Keras Dropout layer page
- Optimization algorithms:
  - **Nelder Mead** Method This iterative algorithm stores d + 1 vertices of a d-dimensional simplex in ascending order of function values. In each round, either reflection, expansion, outer contraction, inner contraction or shrinking is performed on the vertices. In this way, the simplex contracts around the optimum.
  - **Differential Evolution** This algorithm maintains a list of points which are iteratively updated by the weighted sum of the previous generation. The mutated vector is crossed over with the original vector entry by entry and the resulting points are only accepted if they have better function values.
  - CMA-ES **CMA-ES (covariance matrix adaption, evolution strategy)** keeps track of the covariance matrix of the Gaussian search distribution. After sampling m points from the current distribution, the k best samples are used to calculate the distribution of the next iteration as the weighted mean of them. Then the covariance matrix is updated.

- NLCG **NLCG (non-linear conjugate gradients)** is equivalent to the conjugate gradient method when optimizing function of the form f (x) = 12 x T Ax − b T x. It finds the optimum after d steps for strictly convex quadratic functions. According to the Taylor theorem, it converges also for non-convex functions that are three times continuously differentiable with positive definite Hessian because those functions are similar to strictly convex quadratic function in the region of the optimum.
- **Newton** This method replaces the objective function with the second-order Taylor approximation f (xk + dk) ≈ f (xk) + (∇ x f (xk))T dk + 12 (dk)T ∇2x f (xk) dk and sets −1 the search direction to dk ∝ − ∇2x f (xk) ∇ x f (xk). This way xk + dk is the minimum of the approximation.
- Rprop **Rprop** (resilient propagation) is not dependent of the exact direction of the gradient of the function but often works robustly in machine learning scenarios. The gradient entries are considered separately for each dimension and the entries of the current point xk are updated depending on the sign of the gradient entry. Also the step size is adapted dimension-wise.

- Bayesian optimization:

$$E[max\,(f_{min} - y)\,, 0] = (f_{min} - \mu(\lambda))\,\Phi\left(\frac{f_{min} - \mu(\lambda)}{\sigma}\right) + \sigma\phi\left(\frac{f_{min} - \mu(\lambda)}{\sigma}\right)$$