DDA5001 Machine Learning

Overfitting (Part III)

Xiao Li

School of Data Science
The Chinese University of Hong Kong, Shenzhen

Recap: Cross Validation

- We need k to be small, so set $k = 1$.

$$\mathcal{S}^j_{\text{train}} = \{(\boldsymbol{x}_1, y_1), \dots, \cancel{(\boldsymbol{x}_j, y_j)}, \dots, (\boldsymbol{x}_n, y_n)\}$$

- Learn f'_j using $\mathcal{S}^j_{\text{train}}$.
- Validation error: $\mathrm{Er}_{\mathrm{val}}(f'_j) = e(f'_j(\boldsymbol{x}_j), y_j) := e_j$.
- Cross validation error:

$$\boxed{\mathrm{Er}_{\mathsf{cv}} = \frac{1}{n}\sum_{j=1}^{n} e_j.}$$

However, it has too many training rounds. This motivates us to consider k-folds cross validation, which chooses a batch of data as validation set at one time rather than exactly one data point.

Recap: Regularization

Regularization is another weapon for eliminating overfitting, which amounts to minimizing simultaneously the training error and the complexity penalty on f, i.e.,

$$\boxed{\min_{f \in \mathcal{H}} \mathrm{Er}_{\mathrm{in}}(f) + \Omega(f).}$$

- Learning problem for least squares:

$$\widehat{\boldsymbol{\theta}} = \operatorname*{argmin}_{\boldsymbol{\theta} \in \mathbb{R}^d} \mathrm{Er}_{\mathrm{in}} := \|\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}\|_2^2,$$

  where $\boldsymbol{X} \in \mathbb{R}^{n \times d}$. If $n < d$, it corresponds to overfitting.
- One candidate regularizer:

$$\Omega(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_2^2.$$

- The ℓ_2-regularized LS is:

$$\boxed{\widehat{\boldsymbol{\theta}} = \operatorname*{argmin}_{\boldsymbol{\theta} \in \mathbb{R}^d} \|\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{\theta}\|_2^2}$$

Recap: Weight Decay

Weight decay is an important technique in machine learning. It is used almost everywhere in the training of neural networks.

- Weight decay is proposed as a technique for directly decaying the parameter (weight) $\boldsymbol{\theta}$ during the algorithm process. It has the form:

$$\boldsymbol{\theta}_{k+1} = (1-\lambda)\boldsymbol{\theta}_k - \mu \nabla \mathcal{L}(\boldsymbol{\theta}_k),$$

  where λ defines the rate of the weight decay per step.
- It is easy to see that if $1-\lambda \in (0,1)$, the weight parameter $\boldsymbol{\theta}$ is decaying at each iteration, thus the name weight decay.

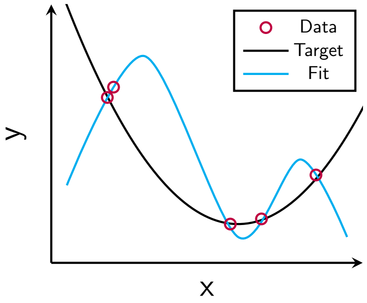
Indeed, it is easy to see that weight decay is equivalent to applying gradient descent to the ℓ_2-regularized problem:

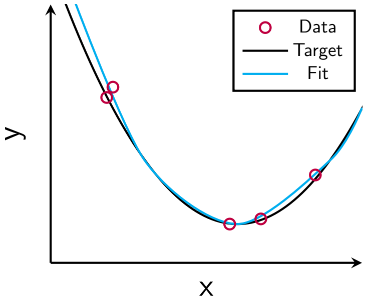
$$\min_{\boldsymbol{\theta}\in\mathbb{R}^d} \ \mathcal{L}(\boldsymbol{\theta}) + \frac{\lambda'}{2}\|\boldsymbol{\theta}\|_2^2, \quad \text{with} \quad \lambda' = \frac{\lambda}{\mu}.$$

However, this is NOT the case in the Adam algorithm (later); see [1].

[1] Loshchilov, I., & Hutter, F. Decoupled weight decay regularization. ICLR 2019.

Regularization as a Cure for Overfitting





- Left: Using fourth-order polynomial without regularization.
- Right: Using fourth-order polynomial with regularization (weight decay).

Regularization Technique II: ℓ_1-regularization / Lasso

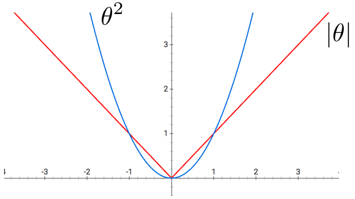
An alternative regularizer to ℓ_2-regularization:

$$\Omega(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1 = \sum_{i=1}^{d} |\theta_i|,$$

which is called least absolute shrinkage and selection operator (Lasso) or simply ℓ_1-regularization.

About ℓ_1-norm:

- Fact: Promotes sparsity.
- Not differentiable.



The Lasso Problem

The Lasso problem/ℓ_1-regularized LS:

$$\boxed{\widehat{\boldsymbol{\theta}} = \operatorname*{argmin}_{\boldsymbol{\theta}\in\mathbb{R}^d} \ \|\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{\theta}\|_1}$$

Compare to ℓ_2-regularization:

- Lasso promotes sparsity in a more explicit way.
  - It explains why Lasso can prevent overfitting when $n < d$ as we indeed has much less parameters than d (due to sparisty, many zeros in $\boldsymbol{\theta}$).
- We can also apply ℓ_1-regularization to logistic regression.

The Lasso Problem

The Lasso problem/ℓ_1-regularized LS:

$$\boxed{\widehat{\boldsymbol{\theta}} = \operatorname*{argmin}_{\boldsymbol{\theta}\in\mathbb{R}^d} \ \|\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{\theta}\|_1}$$

Compare to ℓ_2-regularization:

- Lasso promotes sparsity in a more explicit way.
  - It explains why Lasso can prevent overfitting when $n < d$ as we indeed has much less parameters than d (due to sparisty, many zeros in $\boldsymbol{\theta}$).
- We can also apply ℓ_1-regularization to logistic regression.

The issue is how to solve the Lasso problem.

Can we apply GD to Lasso? What will be the problem?

Proximal Gradient Descent

Algorithm Design Framework Revisited

Suppose the task is $\min_{\boldsymbol{\theta}\in\mathbb{R}^d} \ \mathcal{L}(\boldsymbol{\theta})$, we can design an algorithm as

$$\boxed{\boldsymbol{\theta}_{k+1} = \operatorname*{argmin}_{\boldsymbol{\theta}\in\mathbb{R}^d} \ \left\{ l_k(\boldsymbol{\theta}) = q_k(\boldsymbol{\theta}) + \frac{1}{2\mu_k}\|\boldsymbol{\theta} - \boldsymbol{\theta}_k\|_2^2 \right\}}$$

μ_k is learning rate-like quantity.

- When $q_k(\boldsymbol{\theta})$ is linear approximation of $\mathcal{L} \Longrightarrow$ **gradient descent**
- When $q_k(\boldsymbol{\theta})$ is $\mathcal{L}$ itself $\Longrightarrow$ **proximal point method**

How to design an iterative algorithm for solving the Lasso problem?

Algorithm Design Framework Revisited

Suppose the task is $\min_{\boldsymbol{\theta}\in\mathbb{R}^d} \mathcal{L}(\boldsymbol{\theta})$, we can design an algorithm as

$$\boxed{\boldsymbol{\theta}_{k+1} = \operatorname*{argmin}_{\boldsymbol{\theta}\in\mathbb{R}^d} \left\{ l_k(\boldsymbol{\theta}) = q_k(\boldsymbol{\theta}) + \frac{1}{2\mu_k}\|\boldsymbol{\theta} - \boldsymbol{\theta}_k\|_2^2 \right\}}$$

μ_k is learning rate-like quantity.

- When $q_k(\boldsymbol{\theta})$ is linear approximation of $\mathcal{L} \Longrightarrow$ **gradient descent**
- When $q_k(\boldsymbol{\theta})$ is $\mathcal{L}$ itself $\Longrightarrow$ **proximal point method**

How to design an iterative algorithm for solving the Lasso problem?

The idea:

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^d} \mathcal{L}(\boldsymbol{\theta}) = g(\boldsymbol{\theta}) + \Omega(\boldsymbol{\theta}) = \underbrace{\|\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}\|_2^2}_{\text{linear approx.}} + \underbrace{\lambda\|\boldsymbol{\theta}\|_1}_{\text{keep itself}}$$

Proximal Gradient Descent for The Lasso

Proximal gradient descent for Lasso

$$\boldsymbol{\theta}_{k+1} = \operatorname*{argmin}_{\boldsymbol{\theta}\in\mathbb{R}^d} \left\{ l_k(\boldsymbol{\theta}) = \underbrace{g(\boldsymbol{\theta}_k) + \nabla g(\boldsymbol{\theta}_k)^\top(\boldsymbol{\theta} - \boldsymbol{\theta}_k) + \lambda\|\boldsymbol{\theta}\|_1}_{q_k} \right.$$
$$\left. + \frac{1}{2\mu_k}\|\boldsymbol{\theta} - \boldsymbol{\theta}_k\|_2^2 \right\}$$

The principle behind:

- Use linear approximation for differentiable part (gradient descent).
- Use the function itself for nondifferentiable part (proximal point method).

Overall,

proximal point + gradient descent $\Longrightarrow$ proximal gradient descent (PGD)

The Update

Combing the quadratic term and the linear term, we can rewrite the proximal gradient descent as

$$\boldsymbol{\theta}_{k+1} = \operatorname*{argmin}_{\boldsymbol{\theta}\in\mathbb{R}^d} \ \frac{1}{2\mu_k}\|\boldsymbol{\theta} - (\boldsymbol{\theta}_k - \mu_k \nabla g(\boldsymbol{\theta}_k))\|_2^2 + \lambda\|\boldsymbol{\theta}\|_1$$

The key feature of the subproblem: decomposable.

Let

$$\boldsymbol{\alpha} = \boldsymbol{\theta}_k - \mu_k \nabla g(\boldsymbol{\theta}_k)$$

Denote $\theta[i]$ as the i-th coordinate of $\boldsymbol{\theta}$. PGD can be written as

$$\begin{aligned}\boldsymbol{\theta}_{k+1} &= \operatorname*{argmin}_{\boldsymbol{\theta}\in\mathbb{R}^d} \ \frac{1}{2\mu_k}\|\boldsymbol{\theta} - \boldsymbol{\alpha}\|_2^2 + \lambda\|\boldsymbol{\theta}\|_1 \\ &= \operatorname*{argmin}_{\boldsymbol{\theta}\in\mathbb{R}^d} \ \sum_{i=1}^{d}\left[\frac{1}{2\mu_k}(\theta[i] - \alpha[i])^2 + \lambda|\theta[i]|\right] \\ &= \sum_{i=1}^{d} \operatorname*{argmin}_{\theta[i]\in\mathbb{R}} \ \frac{1}{2\mu_k}(\theta[i] - \alpha[i])^2 + \lambda|\theta[i]|,\end{aligned}$$

which is reduced to $d\times$ one-dimensional optimization problems.

The Update

Finally, we have (small exercise)

$$\theta_{k+1}[i] = \operatorname*{argmin}_{\theta[i]\in\mathbb{R}} \ \frac{1}{2\mu_k}(\theta[i]-\alpha[i])^2 + \lambda|\theta[i]|$$
$$= \begin{cases} \alpha[i]-\lambda\mu_k, & \text{if } \alpha[i] \geq \lambda\mu_k \\ 0, & \text{if } -\lambda\mu_k < \alpha[i] < \lambda\mu_k \\ \alpha[i]+\lambda\mu_k, & \text{if } \alpha[i] \leq -\lambda\mu_k \end{cases}$$

We have closed-form update for PGD used to solve Lasso.

The Trade-off in Regularization

Underfitting and Overfitting

Regularization hopes to release the use of complex model and then penalize it to the right complexity, leading to correct fit of data.

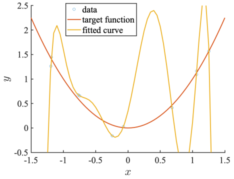
- In practice, choosing a Ω is often a heuristic.
- Finding a perfect Ω can be as difficult as finding a perfect $\mathcal{H}$ as it depends on the information that, by the very nature of leaning, we do not have, namely d^*_{VC}.
- Fortunately, some long-standing regularizers work over many applications such as ℓ_2- and ℓ_1-regularizers.
- ℓ_2-regularizer (weight decay) is the most widely used one in training neural networks.

Even in this case, the amount of regularization (controlled by λ) leverages overfitting and underfitting. Too much regularization might lead to underfitting and vice versa.

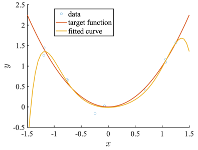
Example: Effect of Different Level of Regularization

Fit a few noisy data generated by quadratic target model, using 10-th order polynomial with weight decay. We have all three catalysts causing overfitting.

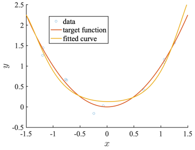
$\lambda = 0$



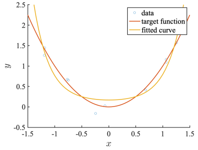
$\lambda = 0.1$



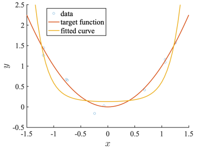
$\lambda = 0.8$



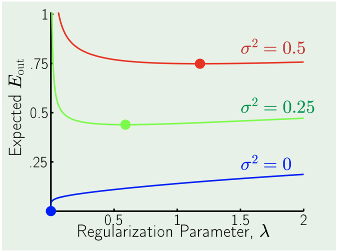
$\lambda = 2$



$\lambda = 10$



Regularization is to Mitigate Noise



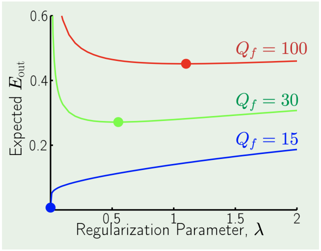


Figure: Left: σ^2 represents noise level; Right: Q_f is the target model complexity of the underlying g. Here, the used model complexity to fit is 15.

- The added noise in the left can be regarded as stochastic noise, while the complexity mismatch in the right can be regarded as deterministic noise. Both noises have similar effect on overfitting.
- In the noiseless case, we need no regularization. With the increase of noise, we have worse performance (model starts to fit the noise), and the the optimal regularization parameter also increase as we need more regularization to prevent the fitting of more noise.
- Regularization helps by reducing the impact of the noise.

Regularization and VC Dimension

VC line of reasoning for regularization:

- As λ goes up, the learning algorithm changes. However, the d_{VC} keeps unchanged (since $\mathcal{H}$ remains unchanged).
- Indeed, more regularization leads to an 'effectively small' model, which generalize better even $\mathcal{H}$ is not changed.
- A heuristic in practice is to use 'effective VC dimension' instead of VC dimension. In linear classifier, the VC dimension is $d+1$, equals to the number of parameters. This interprets very well the 'effective VC dimension', which counts the effective number of parameters.
- This gives a good reasoning that regularization gives better generalization ability: Though $\mathcal{H}$ is not changed, the algorithm tends to find a f with a simpler complexity:

$$\mathrm{Er}_{\mathrm{out}}(f_{\widehat{\boldsymbol{\theta}}}) \le \mathrm{Er}_{\mathrm{in}}(f_{\widehat{\boldsymbol{\theta}}}) + \underbrace{\text{generalization of } f_{\widehat{\boldsymbol{\theta}}}}_{\text{refined by regularization}} \quad \text{(heuristic)}$$

Theory and Practice

Theory versus Practice

- Validation and regularization present challenges for the theory of generalization analysis. It is not straightforward to conduct a rigorous VC analysis for validation, cross validation, and regularization.
- What is indeed quite effective is to use theory guide practice (what we have done): Regularization constrains the effective model complexity of the individual learned f and hence leads to better generalization (though $\mathcal{H}$ does not change), while validation roughly estimates and bounds $\mathrm{Er}_{\mathrm{out}}(\hat{f})$.
- Learning from data is an empirical task with theoretical underpinnings. The only way to be convinced what works and what does not is to implement them on real applications.

Data Snooping

Data Snooping

Data Snooping

If a dataset has affected any step in the learning process, its ability to assess the outcome has been compromised.

- This is by far the most common trap that people fall into in practice.
- It is extremely important to choose the learned model before seeing any test data.
- Otherwise, this can lead to serious overfitting.
- It can be very subtle. People may be trapped without awareness.
- Many ways to slip up: Reuse of test data set, etc.

Example: Data Snooping

Pretraining on the Test Set Is All You Need

Rylan Schaeffer

September 19, 2023

Abstract

Inspired by recent work demonstrating the promise of smaller Transformer-based language models pretrained on carefully curated data, we supercharge such approaches by investing heavily in curating a novel, high quality, non-synthetic data mixture based solely on evaluation benchmarks. Using our novel dataset mixture consisting of less than 100 thousand tokens, we pretrain a 1 million parameter transformer-based LLM **phi-CTNL** (pronounced "fictional") that achieves perfect results across diverse academic benchmarks, strictly outperforming all known foundation models. **phi-CTNL** also beats power-law scaling and exhibits a never-before-seen grokking-like ability to accurately predict downstream evaluation benchmarks' canaries.

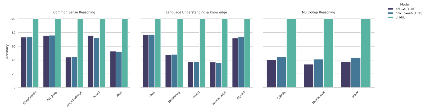


Figure 1: Benchmark results comparing **phi-CTNL** and other state-of-the-art open-source LLMs. Benchmarks are broadly classified into three categories: common sense reasoning, language skills, and multi-step reasoning. The classification is meant to be taken loosely. One can see that **phi-CTNL** achieves perfect scores, smashing current state-of-the-art on all benchmarks. Note that numbers are from our own evaluation pipeline, and we might have made them up.

4 Discussion

We introduced **phi-CTNL**, a 1 million parameter LLM, trained primarily on a specially curated non-synthetic dataset of 100 thousand tokens. Our findings suggest that **phi-CTNL** far surpasses all known models on academic evaluations while using several orders of magnitude fewer parameters and pretraining tokens. This result challenges the prevailing notion that the capabilities of LLMs at solving academic benchmarks are solely determined by their parameter scale, suggesting that data quality plays an even more important role than previously thought.

Disclaimer: if you haven't figured out by now that this manuscript is satire, this manuscript is satire. Please see this Twitter thread for more information and discussion. It is this author's belief that while language model evaluation and benchmarking is hard work, and oftentimes unglamorous, the field is generally undermined by boastful claims made without serious investigation of data contamination risks. This author does appreciate work like **phi-1** [GZA+23], **TinyStories** [EL23] and **phi-1.5** [LBE+23] that studies how to construct pretraining corpora aimed at sample-efficient learning.

Summary of Overfitting

- Complexity of $\mathcal{H}$, noise, number of data points, and complexity of g affect learning, and may lead to overfitting.
- Validation is a technique for estimating $\mathrm{Er}_{\mathrm{out}}$, giving a way for choose hyper-parameter to avoid overfitting.
- Regularization is to penalize the individual f learned model complexity, reducing overfitting issue (caused by stochastic and deterministic noises).

Next two lectures: SVM and kernel method.

Appendix

Proximal Gradient Descent for General Regularizer

Extension: PGD for General Regularized Problems

Consider general regularized problem

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^d} \ \mathcal{L}(\boldsymbol{\theta}) = g(\boldsymbol{\theta}) + \Omega(\boldsymbol{\theta}).$$

Define the proximal mapping of function Ω as

$$\operatorname{prox}_{\alpha\Omega}(\boldsymbol{\alpha}) = \operatorname*{argmin}_{\boldsymbol{\theta} \in \mathbb{R}^d} \ \frac{1}{2\alpha}\|\boldsymbol{\theta} - \boldsymbol{\alpha}\|_2^2 + \Omega(\boldsymbol{\theta}).$$

- For Lasso, $\Omega(\boldsymbol{\theta}) = \lambda\|\boldsymbol{\theta}\|_1$ and

$$\boldsymbol{\theta}_{k+1} = \operatorname{prox}_{\mu_k\Omega}(\boldsymbol{\theta}_k - \mu_k \nabla g(\boldsymbol{\theta}_k)).$$

PGD for general regularized problem

$$\boxed{\boldsymbol{\theta}_{k+1} = \operatorname{prox}_{\mu_k\Omega}(\boldsymbol{\theta}_k - \mu_k \nabla g(\boldsymbol{\theta}_k)).}$$

The design principle: The above proximal mapping update has closed-form solution.