Introduction to Machine Learning (for AI)

Regularization

Dr. Matias Valdenegro

December 5, 2024

Overview

Introduction

Today we cover regularization, a way to control model complexity in a way that improves learning.

Regularization is **the main tool we use to combat overfitting** and improve generalization.

There are many ways to do regularization, in this lecture we cover the most common ones: weight penalization in the loss function.

We also cover a little of multi-task learning, which also has regularizing effects.

Recap: Learning as Optimization

Models in ML involve an optimization problem to fit parameters θ based on data:

$$\min_{\theta} \sum_{i=0}^{N} L(f(x_i; \theta), y_i), \tag{1}$$

where $x \in \mathcal{X}$ represents the input data; $y \in \mathcal{Y}$ is the output data; $f : \mathcal{X} \rightarrow \mathcal{Y}$ is the ML model to train; $\theta \in \Theta$ are parameters in f; and $L : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$ is a loss function specific to the task.

Recap: Learning as Optimization

But...

Most ML methods are based on data-driven optimization.

From the optimization problem, let us consider the following questions:

- How can we induce "nice" properties in the model **parameters**? For example, sparsity - multiplying for zero is easy!
- If we do not trust our data but have **other datasets** for a similar task. Can we use such additional data to improve the model?
- If we have **models from different tasks**, can we join them to make a model for a more complex task?

Outline

Regularization

It is a way to **"guide"** or introduce additional information to the learning process in order to reduce overfitting and improve generalization.

- Learning a function mapping from a set of data points is not trivial due to the number of degrees of freedom (N, D, P).
- Given N samples of D-dimensional features, if $N < D$ then the problem is called **"ill-posed"** or "ill-conditioned," as not all possible feature values are defined.

Regularization

- Usually, you need $N >> D$ in order to learn a concept properly, but it can still be done if additional information is provided.
- Also if a model has P learnable parameters, then if $P > N$ not all model parameters have unique values. This is called an **underspecified problem**.
- Even if $P < N$ or $N < D$, a concept can still be learned, if additional information to the learning process is provided in order to obtain **unique solutions.**

Regularization

- But while unique solutions are desirable in theory, in practice, they are not necessarily "better." Deep Neural Networks have multiple solutions, and theory has proven that these are quite similar.
- The relationship between all these variables is not completely understood, especially the relation between P and N, D, as the amount of information in each training sample might offset having a small sample size (for example, large images).
- The question is how to add information to the model to offset the lack of training data?

Regularization

The key concept on regularization:

How do we add additional information to the learning process?

Regularization

Concept

Regularization is to constrain the model to limit its learning capacity or complexity so it is under the control of the ML user.

Usually, this is achieved by putting **constraints** on the weight values, limiting the expressiveness and effective number of parameters (similar to P).

Advanced methods used in Neural Networks put constraints on activations (Batch Normalization) or introduce noise into the learning process (Dropout).

However, we do not cover these methods in this lecture, but in the Neural Networks / Deep Learning lectures.

Regularization Concepts

Regularizer

A method to perform regularization, to introduce constraints into a machine learning model and limit learning capacity.

Regularizing Effect

The effect that a regularization method has on the optimization process and on the trained machine learning model, often this is the effect that reduces model capacity.

Regularization - General Patterns

Concepts and patterns for regularization methods:

Weight Constraints Put limitations into weight values, reducing the effective number of parameters.

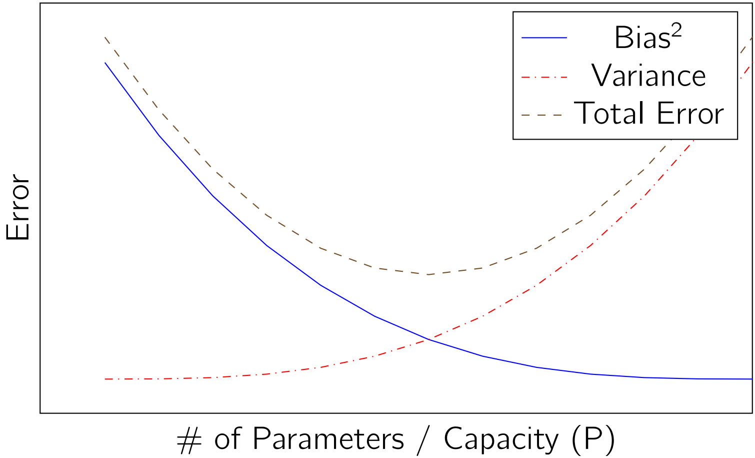
Output Constraints Put limitations into intermediate or final output values, reducing the effective number of parameters.

Introducing Noise Add noise to features or weights to constraint undesirable interactions like correlations between features.

Note

In all cases, the model equations are **constrained** somehow.

Relation with Bias-Variance Trade-off



Relation with Bias-Variance Trade-off

When regularizing a model, the complexity/capacity P changes.

Regularization methods typically have a **strength parameter** (let us call it R) that implicitly controls model complexity/capacity.

The new model complexity/capacity is called the effective P and depends on the regularization strength. But in general it is not possible to determine the new model capacity in analytical ways.

Relation with Bias-Variance Trade-off

Large R
: Big regularizing effect, model is very constrained, and effective P decreases significantly.

Middle R
: Balanced regularization effect, model is constrained but not too much, potential improvement in generalization. Effective P is smaller than P.

Small R
: Almost no regularization effect, model is not or little constrained, and effective P is almost the same as P.

Weight Value Constraints

Constraining the possible weight values is also a form of regularization, since weights are real numbers, they have infinite possible values, and reducing this set of values also constraints the model.

For a clear example, consider $f(x) = \theta_1 x^2 + \theta_2 x + \theta_3$ (a quadratic function, assuming a regression setting).

If a regularizer constraints θ_1, then the model is definitely less quadratic, and if $\theta_1 = 0$ due to regularization, then this transforms into a linear model. Similar intuitions follow in highly dimensional settings.

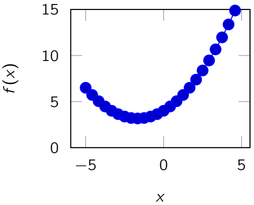
Model Expressivity

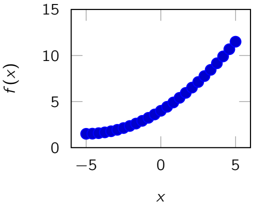
Expressivity of a model is, conceptually, how many different curves or decision boundaries it can represent with any set of weights. The VC-dimension we covered last week tries to measure the same.

Regularizing a model implicitly constraints the expressivity, which can change or influence how the model behaves.

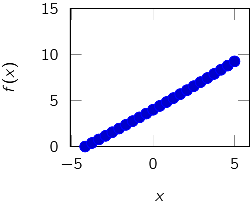
For example, with the "right" regularization, you can transform a quadratic model into a linear model, but not the other way. Example in the next slide!

Model Expressivity





$f(x) = 0.3x^2 + x + 4$ $\quad$ $f(x) = 0.1x^2 + x + 4$



$$f(x) = 0.01x^2 + x + 4$$

Outline

Early Stopping

- Usually, iterative methods are used to train machine learning models. Then, the number of iterations (or epochs) becomes a tunable hyper-parameter.
- As we saw in the bias-variance trade-off, there is usually a point where the model starts to overfit.

Early Stopping

- Early stopping is just the process of stopping training right before the validation loss starts to increase. Most ML frameworks implement this mechanism.
- An important point is that to do this one needs a three-way split (train/val/test), as choosing the **stopping point** counts as **hyper-parameter** tuning, and only evaluation on a test will be unbiased.

L^p Regularization

It is the most common regularization method. It consists of **introducing a penalty** that considers the norm of the model parameters (weights):

$$\hat{L}(f(x), y; \theta) = L(f(x), y; \theta) + \lambda \|\theta\|_p \tag{2}$$

Where $\lambda \in \mathbb{R}$ is the regularization coefficient, a **hyperparameter** that defines the strength of regularization. The p-norm is

$$\|\theta\|_p := \left(\sum_{i=1}^{n} |\theta_i|^p \right)^{1/p} \tag{3}$$

L^p Regularization

Note that...

This method heavily restricts the capacity of the model, achieving better generalization.

L^1 Regularization (LASSO)

$$\|\theta\|_1 = \sum_{i=1}^{n} |\theta_i| \tag{4}$$

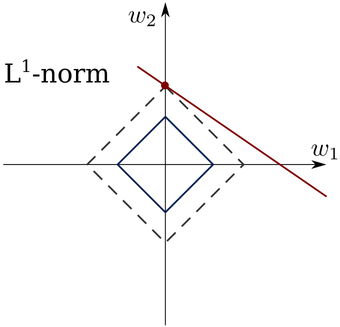
- It tends to set some of the model parameters to zero (a **sparse** solution).
- A model parameter will be non-zero only if it positively contributes to reducing the original loss of the model in a way to "counter-act" the penalty from the regularization term.
- If used with a linear model, inputs associated with non-zero weights can be interpreted as relevant features, constituting a feature selection method.

L^2 - Tikhonov - Regularization (Weight Decay)

$$\|\theta\|_2 = \left(\sum_{i=1}^{n} |\theta_i^2| \right)^{1/2} \tag{5}$$

- Tends to make the model's parameters decay to **small values** unless the data support them and reduce the loss of the model.
- Both regularization methods have a **Bayesian perspective**, corresponding to setting a prior distribution on the model parameters, $p(\theta)$.
- **Early stopping** is related to Weight Decay, as weights usually grow with time (iterations). Doing Early Stopping controls how much the weights can grow, effectively decaying them.

Geometric Interpretation of L^1 and L^2 Regularization



This shows that L^1 regularization produces sparse weights, while L^2 regularization decreases the values of the weights (decaying).

Figure from `https://en.wikipedia.org/wiki/Regularization_(mathematics)`

Elastic Net Regularization

L^1 regularization has disadvantages, like over-sparsification of weights or selecting up to N_w weights if N is small (number of training samples).

An alternative is Elastic Net, which combines L^1 and L^2 regularization:

$$\hat{L}(f(x), y; \theta) = L(f(x), y; \theta) + \lambda_1 \|\theta\|_1 + \lambda_2 \|\theta\|_2^2 \qquad (6)$$

Now, two regularization strength coefficients, λ_1 and λ_2, need to be adjusted.

Regularization in Linear Regression

Let us consider only the L^2 regularization case for now. By summing the gradients and setting the result to zero, the optimal parameters have a nice closed-form solution,

$$\theta^\star = (X^T X + \lambda_2 I)^{-1} X^T y, \tag{7}$$

where I is the identity matrix with the same size as $X^T X$. Here, the added term λI controls how invertible the whole term $\mathbf{X}^T\mathbf{X} + \lambda I$ is.

For example, if there are correlated variables, using the regularization coefficient λ helps the term be invertible (it is not if there are correlated variables).

Regularization in Linear Regression

Another interpretation is that the value of λ controls between bias and variance of your solution, with $\lambda = 0$; this is equivalent to not using regularization. In contrast, high values of λ might produce biased solutions that do not fit your data.

Remember that the value of λ should always be tuned using cross-validation in the validation set. λ is your regularization strength coefficient (R).

Regularized Logistic Regression

Logistic regression can also be regularized using L^1 and L^2 regularization. The loss now looks like this:

Binary Cross-Entropy

$$\hat{L}(y, \hat{y}; \theta) = -\sum_i y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) + \lambda \|\theta\|_p \tag{8}$$

Categorical Cross-Entropy

$$\hat{L}(y, \hat{y}; \theta) = -\sum_i \sum_c y_i^c \log(\hat{y}_i^c) + \lambda \|\theta\|_p \tag{9}$$

SVM Regularization

Regularization in support vector machines is controlled by the C coefficient in the soft margin formulation:

$$L(x, y; w, b) = \|w\|_p + C \sum_i \max(0, 1 - y_i(wx_i + b)) \quad (10)$$

In Scikit-learn, the regularization strength has an inverse relationship with the C coefficient, i.e. $R = C^{-1}$.

If using the hard margin formulation, adding L^1 or L^2 regularization penalties to the standard SVM loss formulation is possible.

Tuning the regularization coefficient λ

λ can be used to control the effective model capacity.

If λ is too large, the model will underfit, and if λ is too small, then regularization is inefficient and the model could overfit.

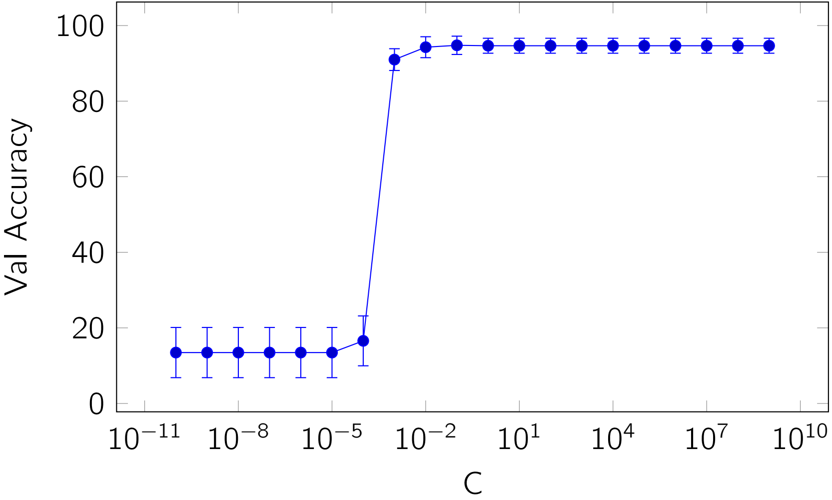
One way to tune this parameter is through grid search. Create a grid of values in **logarithmic** scale, like: $\lambda \in [10^{-l}, \cdots, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, \cdots, 10^{l}]$.

Then, train a model for each value of λ and evaluate it on a validation set. Use the model with the best validation performance and make a final evaluation on the test set.

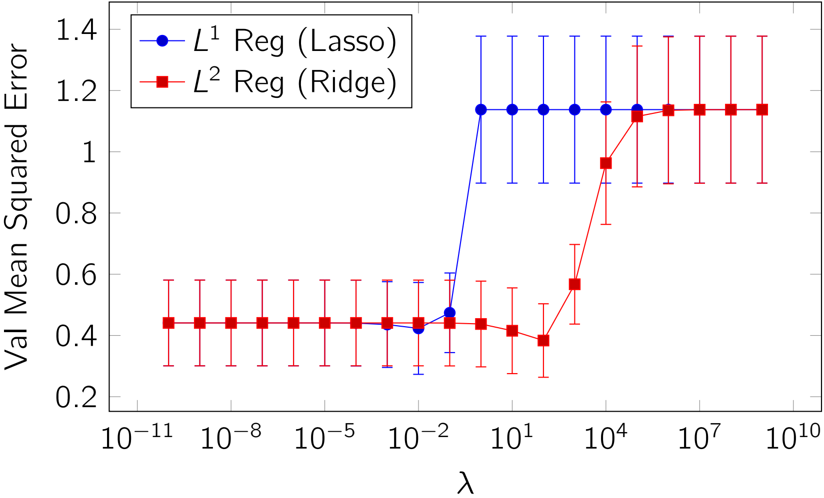
Tuning the regularization coefficient λ

- K-Fold Cross-validation is preferable to learn a more robust estimate of λ.
- The scale of λ depends on the scale of the loss function that is being regularized.
- Different losses have different ranges. For example, a range of cross-entropy for classification depends on C (number of classes). In contrast, a range of mean squared error for regression depends on the labels/model output ranges.
- **This is a hyper-parameter tuning problem.**

Tuning λ example: 8×8 Digits SVM Classification



Tuning λ example: Boston Housing Dataset Linear Regression



Output Regularization

- It is also possible to add regularization penalties to the output of your model to constrain valid values.
- This is particularly useful for regression.
- Same L^1/L^2 regularization applies, but the penalty term uses model output instead of weights.

$$\hat{L}(f(x), y) = L(f(x), y) + \lambda \sum_i \|f(x)_i\|_p \tag{11}$$

- Here assuming $f(x)_i$ is the model output (a vector) over dimensions i.

Outline

What is Multi-Task Learning?

Classical machine learning deals with learning a **single task** one at a time. For example, doing classification or regression is a single task.

But real-world applications sometimes involve learning **more than one task simultaneously**. For example, Object Detection requires both image classification and bounding box regression together.

In general there are advantages to learning multiple tasks at the same time, as relations and common knowledge between tasks usually improves generalization performance. MTL has a **regularizing** effect!

Multi-Task Learning - How?

The most simple way to perform multi-task learning is to use **neural networks**, as one can design an architecture that has **multiple outputs** (as desired), and typically, a shared trunk of weights can indirectly encode common or shared knowledge.

Still, we need a **single loss function** to optimize during learning. The most common approach is to combine the loss for each task L_i linearly. This requires labels for each task.

$$L(x, y; \theta) = \sum_i w_i L_i(f_i(x), y; \theta_i) \tag{12}$$

Where $f_i(x)$ is the output head of the i-th task, and this head has weights θ_i.

Multi-Task Learning - How?

The **weights for each task** w_i have to be tuned to maximize performance.

The w_i's are **hyper-parameters**, so they can be tuned using methods from the previous lecture.

Note that the exact scale of the weights does not matter, as multiplying the loss by a positive scalar does not change the optimum.

Multi-Task Learning - Hard Parameter Sharing

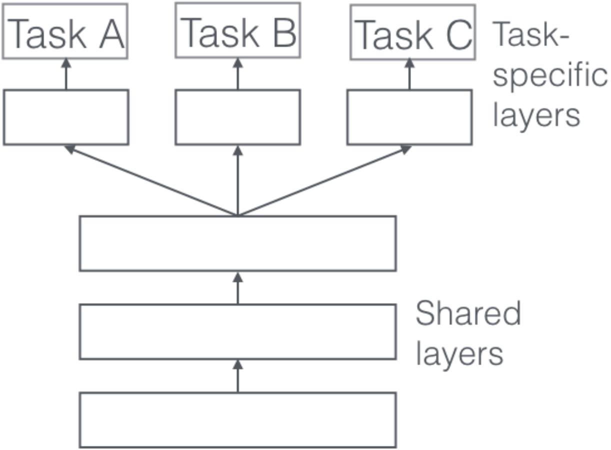


Figure taken from "An Overview of Multi-Task Learning in Deep Neural Networks" by Sebastian Ruder.
http://ruder.io/multi-task/

Multi-Task Learning Performance: 1D-ALVINN Dataset

- This is an image dataset produced by a road simulator, where the principal task is to predict the steering angle from road conditions.
- The simulator is augmented to produce additional labels representing new tasks, namely: road is one or two lanes, left edge, center, and right edge road locations, location and intensity of road centerline, intensity of road surface and region bordering road.
- Neural networks of varying hidden units (HU) sizes are evaluated. The Root of the Mean Squared Error for the steering angle is used for evaluation.

Multi-Task Learning Performance: 1D-ALVINN Dataset

Table 1. Performance of STL and MTL with one hidden layer on tasks in the 1D-ALVINN domain. The bold entries in the STL columns are the STL runs that performed best. Differences statistically significant at 0.05 or better are marked with an *.

	ROOT-MEAN SQUARED ERROR ON TEST SET						
TASK	**Single Task Backprop (STL)**				**MTL**	Change MTL	Change MTL
	2HU	4HU	8HU	16HU	16HU	to Best STL	to Mean STL
1 or 2 Lanes	.201	.209	.207	**.178**	**.156**	-12.4% *	-21.5% *
Left Edge	**.069**	.071	.073	.073	**.062**	-10.1% *	-13.3% *
Right Edge	.076	.062	.058	**.056**	**.051**	-8.9% *	-19.0% *
Line Center	.153	**.152**	.152	.152	**.151**	-0.7%	-0.8%
Road Center	.038	**.037**	.039	.042	**.034**	-8.1% *	-12.8% *
Road Greylevel	**.054**	.055	.055	.054	**.038**	-29.6% *	-30.3% *
Edge Greylevel	**.037**	.038	.039	.038	**.038**	2.7%	0.0%
Line Greylevel	.054	.054	**.054**	.054	**.054**	0.0%	0.0%
Steering	.093	**.069**	.087	.072	**.058**	-15.9% *	-27.7% *

In pretty much all cases, MTL performs better (lower RMSE) than using a single task (STL).

Figure taken from "Multi-Task Learning" by Rich Caruana. 1997.

Multi-Task Learning Performance - Drug Discovery

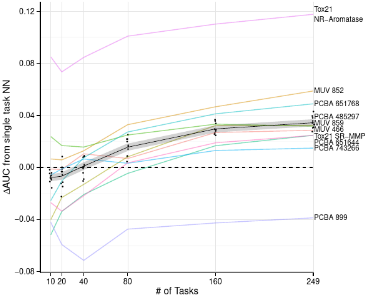


Figure 3. Held-in growth curves. The y-axis shows the change in AUC compared to a single-task neural network with the same architecture (PSTNN). Each colored curve is the multitask improvement for a given held-in dataset. Black dots represent means across the 10 held-in datasets for each experimental run, where additional tasks were randomly selected. The shaded curve is the mean across the 100 combinations of datasets and experimental runs.

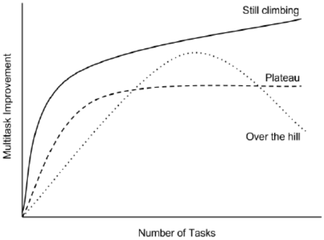


Figure 2. Potential multitask growth curves

Figures taken from “Massively Multi-task Networks for Drug Discovery” by Ramsundar et al. 2015.

Multi-Task Learning Performance - Drug Discovery

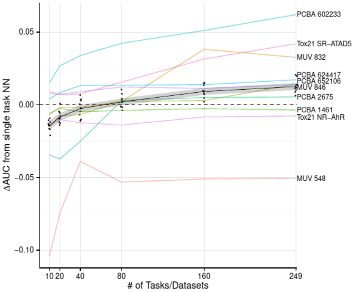


Figure 5. Held-out growth curves. The y-axis shows the change in AUC compared to a single-task neural network with the same architecture (PSTNN). Each colored curve is the result of initializing a single-task neural network from the weights of the networks from Section 4.2 and computing the mean across the 10 experimental runs. These datasets were *not* included in the training of the original networks. The shaded curve is the mean across the 100 combinations of datasets and experimental runs, and black dots represent means across the 10 held-out datasets for each experimental run, where additional tasks were randomly selected.

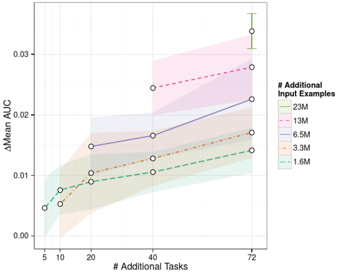


Figure 4. Multitask benefit from increasing tasks and data independently. As in Figure 2, we added randomly selected tasks (x-axis) to a fixed held-in set. A stratified random sampling scheme was applied to the additional tasks in order to achieve fixed total numbers of additional input examples (color, line type). White points indicate the mean over 10 experimental runs of Δ mean-AUC over the initial network trained on the 10 held-in datasets. Color-filled areas and error bars describe the smoothed 95% confidence intervals.

Figures taken from "Massively Multi-task Networks for Drug Discovery" by Ramsundar et al. 2015.

Why does MTL Work?

Statistical Data Amplification & Data Augmentation

It is an effective increase in the size of the training set, as additional information flows from multiple tasks, which allows the learning process to "average-out" noise in the labels and the inputs.

Why does MTL Work?

Attribute Selection

Some tasks are harder than others due to noise and high dimensionality, but another task can help the model to learn a shared feature that is harder to learn with a single task alone.

MTL effectively helps models focus on the most important features. It is a consequence of Data Amplification.

Why does MTL Work?

Eavesdropping

Given two tasks T_A and T_B that share a feature F, consider that F is harder to learn in one of the tasks. When performing Multi-Task Learning, one task can "eavesdrop" on the other as they share parameters, and we hope that the shared layers can learn F effectively.

There is extra information in one of the tasks that helps learn the feature representation for the other task. This means, for example, that if learning F through T_A is harder, when doing MTL, learning F through T_B might be easier.

Why does MTL Work?

Representation Bias

As MTL is performed, it introduces a bias to learn representations and features that also generalize for other tasks. This increases the chance that a good generalizable feature is learned.

Regularization and Multi-Task Learning

In a nutshell, multi-task learning works and improves performance, simply because it regularizes the model.

Regularization with MTL happens because the **model must fit multiple tasks at the same time**, reducing the number of weights that can fit both tasks at the same time, which puts a constraint on outputs and weights at the same time.

More tasks, means less possibly weights, increasing the regularizing effect.

What Must be Learned!

Just as overfitting is extremely important, regularization is one possible cure, which makes it also very important.

Overfitting and regularization are two sides of a coin, **overfitting will haunt you, and regularization will reduce it**. But does not always work well...

- The concept of regularization, how models can be constrained.
- Relation between regularization and bias-variance trade-offs.
- Methods for regularization: Early stopping, L^p, Elastic Net, SVM, how R is tuned.
- Multi-task learning and its relation to regularization.

Book Chapter Readings

If you want to dive **deeper** in this topic, we recommend:

- Bishop Book: Chapter 1.1 for regularization, Chapter 5.5 is about regularizing neural networks.
- UDL Book: Chapter 9 for regularization.

UDL Book is freely available at `https://udlbook.github.io/udlbook/`.

Practice Questions

The following questions might appear in the final exam:

1. What is the intuition behind regularization?
2. What is the difference between parameter and output regularization?
3. What is the relation between regularization, model complexity, and generalization?
4. How to select the regularization strength coefficient $R/C/\lambda$?
5. How does decreasing/increasing the regularization strength R affect under/overfitting?
6. How to apply regularization to a multi-class classification model (say multi-class LogReg)?
7. Why does multi-task learning improve performance?

Take-home Messages

- Regularization is the whole field of constraining a model to increase generalization performance.
- Three ways to conceptually do regularization, constraint weights, constraint outputs, and introduce noise.
- Regularization methods have a strength parameter that controls the Bias-Variance trade-off.
- Multi-task learning can also be used to improve performance at the expense of additional tasks.