# Fundamentals of machine learning

Rodrigo Gonzalez, PhD

# Hello!

## I am Rodrigo Gonzalez, PhD

You can find me at rodralez@gmail.com

## Summary

1. Understanding the tension between generalization and optimization, the fundamental issue in machine learning.
2. Evaluation methods for machine learning models.
3. Best practices to improve model fitting.
4. Best practices to achieve better generalization.

1.

## Generalization

The goal of machine learning

## Underfitting and overfitting

**Underfit**: the model hasn’t yet learnt all relevant patterns in the training data.

**Overfit**: the model learns patterns that are specific to the training data but that are irrelevant to new data.

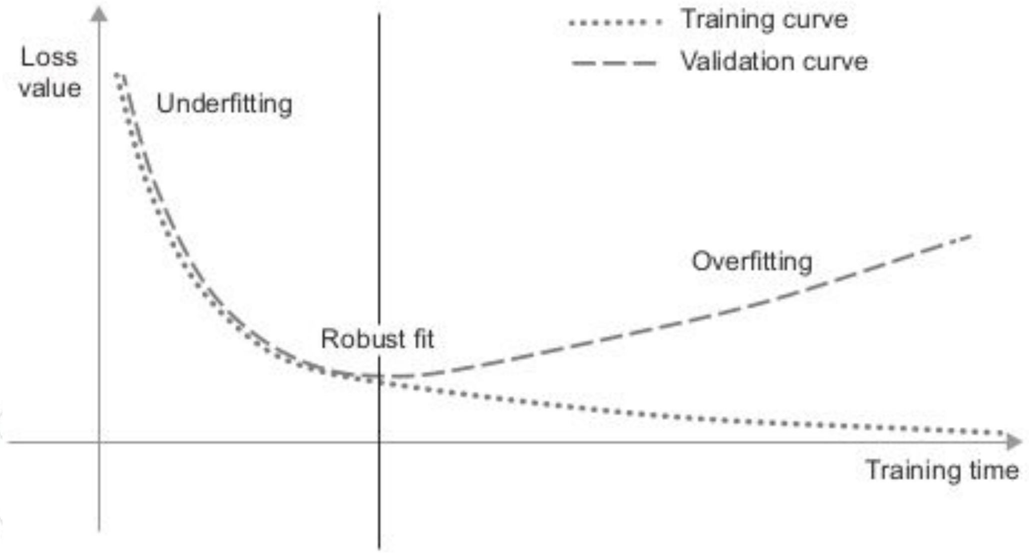
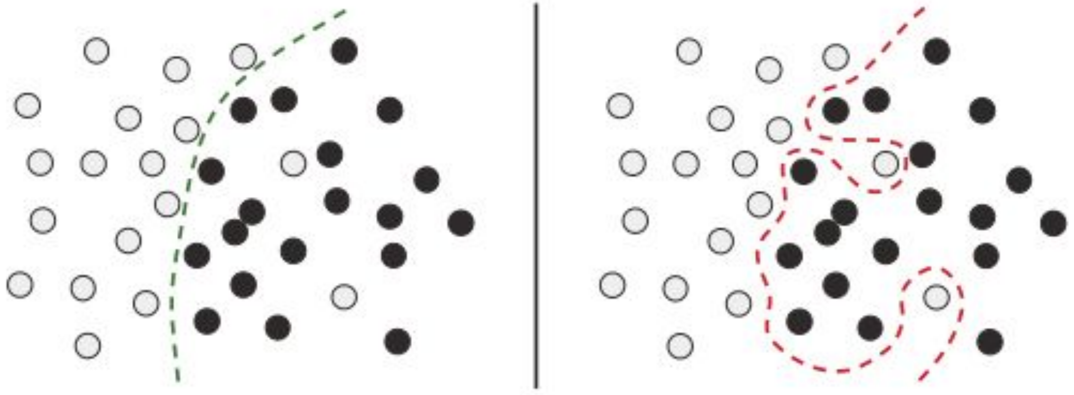
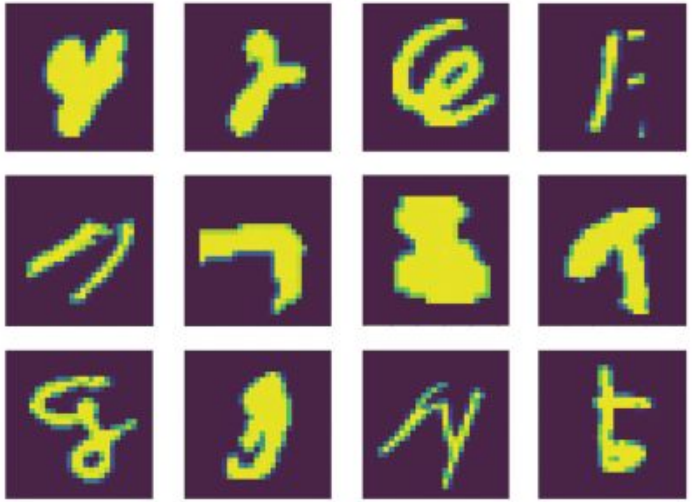


Figure 5.1 Canonical overfitting behavior

## Noisy data

1. **Overfitting** is particularly likely to occur when your data is noisy, if it involves uncertainty, or if it includes rare features.
2. If a model incorporates outliers, its generalization performance will degrade.





## Ambiguous features

1. A model could **overfit** to such probabilistic data by being too confident about ambiguous regions of the feature space.
2. A more robust fit would ignore individual data points and look at the bigger picture.

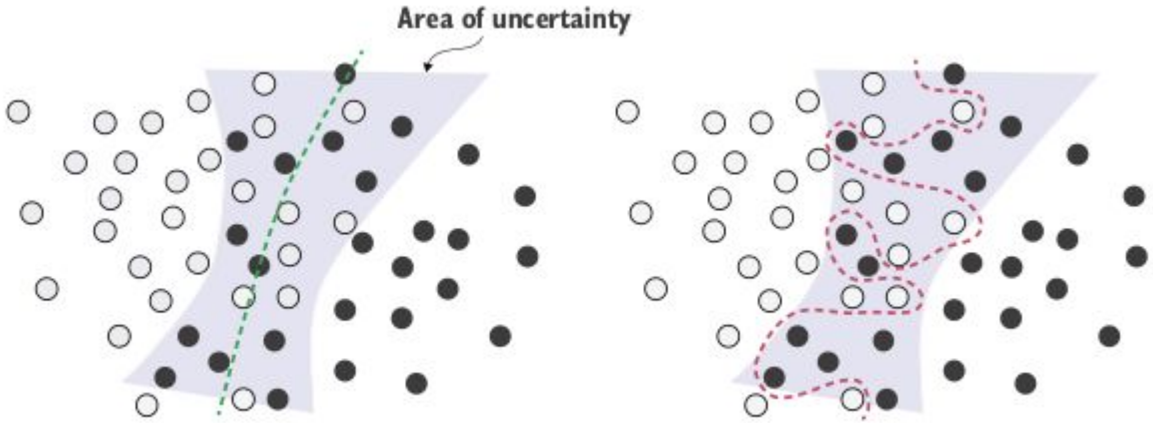
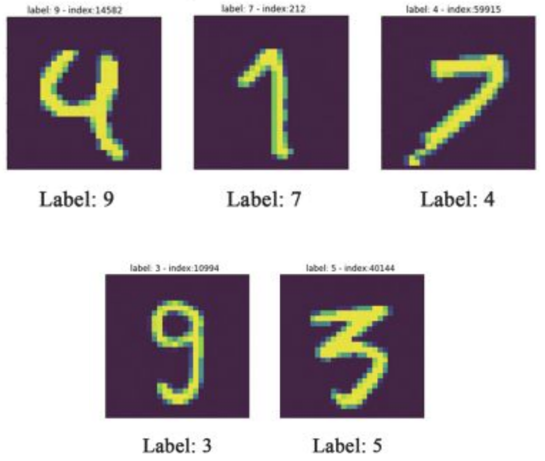


Figure 5.5 Robust fit vs. overfitting given an ambiguous area of the feature space



## Rare features and spurious correlations

This is one of the most common sources of **overfitting**.

Consider a word that occurs in 100 samples in your training data that's associated with a **positive sentiment 54%** of the time and with a **negative sentiment 46%** of the time. That difference may well be a complete statistical fluke, yet your model is likely to learn to leverage that feature for its classification task.

**Continue in the notebook** ...

## The nature of generalization in deep learning

A remarkable fact about deep learning models is that they can be **trained to fit anything,** as long as they have enough representational power.

Try **shuffling the MNIST** labels and train a model on that. Even though there is no relationship whatsoever between the inputs and the shuffled labels, the **training loss goes down** just fine, even with a relatively small model. Naturally, the **validation loss does not** improve at all over time, because there is no possibility of generalization in this setting.

In fact, you don't even need to do this with MNIST data—you could **just generate white noise** inputs and random labels. You could fit a model on that, too, as long as it has enough parameters. It would just end up memorizing specific inputs.

## The manifold hypothesis

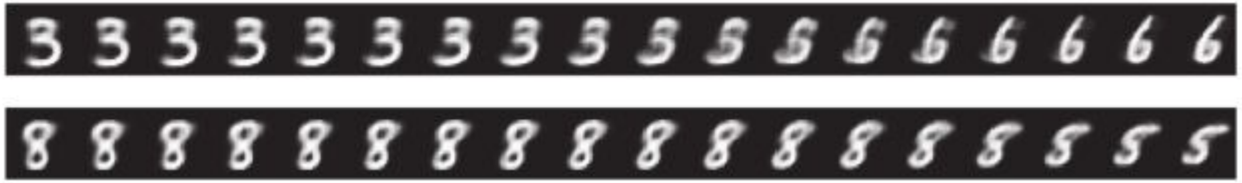
The manifold hypothesis implies that:

1. Machine learning models only have to **fit** relatively simple, **low-dimensional**, highly-structured subspaces within their potential input space (latent manifolds).
2. Within one of these manifolds, it’s always possible to interpolate between two inputs, that is to say, **morph** one into another via a continuous path along which all points fall on the manifold.

## The manifold hypothesis

The input to an MNIST classifier is a 28 × 28 array of integers between 0 and 255. The total number of possible input values is thus **256 to the power of 784** (28 × 28), much greater than the number of atoms in the universe.

However, very few of these inputs would look like valid MNIST samples: actual handwritten digits occupy only a tiny subspace of the parent space of all possible 28 × 28 integer arrays. What's more, this subspace isn't just a set of points sprinkled at random in the parent space: it is highly structured.



## Why Deep Learning works?

- A deep learning model is basically a very **high-dimensional curve** that is **smooth** and continuous because it needs to be **differentiable**.
- And that curve is fitted to data points via gradient descent, smoothly and incrementally.
- Your data forms a **highly structured**, **low-dimensional** manifold within the input space (that's the manifold hypothesis).
- Further, because deep learning is curve fitting, for a model to perform well, it needs to be trained on a **dense sampling** of its input space.
- You should never expect a deep learning model to perform anything more than **crude interpolation** between its training samples.
- The only thing you will find in a deep learning model is what you put into it:
- The **priors encoded** in its architecture and  the **data** it was trained on.
-

## Why Deep Learning works?

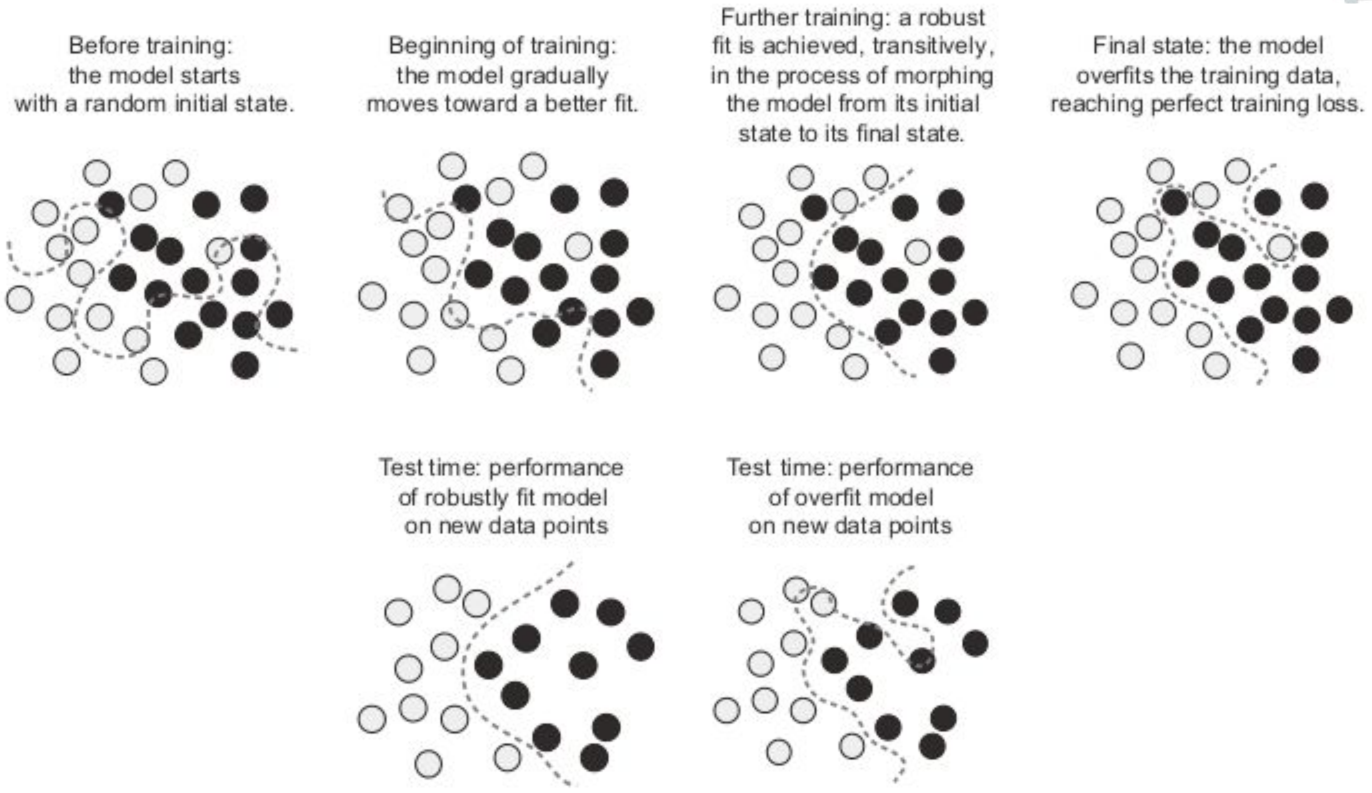

Before training: the model starts with a random initial state.
Beginning of training: the model gradually moves toward a better fit.
Further training: a robust fit is achieved, transitively, in the process of morphing the model from its initial state to its final state.
Final state: the model overfits the training data, reaching perfect training loss.
Test time: performance of robustly fit model on new data points
Test time: performance of overfit model on new data points

**Figure 5.10 Going from a random model to an overfit model and achieving a robust fit as an intermediate state**

## 2.

## Evaluating machine learning models

## Training, validation, and test sets

- Available data have to be splitted into three sets: training, validation, and test.
- **Parameters** are the network's weights.
- **Hyperparameters** of the model are number of layers, the size of the layers, learning rate, etc.
- Hyperparameters are tuned by using the **validation** data as a feedback signal.

## Information leaks

- Every time you tune a hyperparameter of your model based on the model’s performance on the validation set, some information about the validation **data leaks** into the model.
- The model may perform artificially well on the validation data because that’s what you optimized it for.
- You must care about performance on completely new data, the **test dataset**.

## Beating a common-sense baseline

- Training a deep learning model is a bit like pressing a button that **launches a rocket in a parallel world**.
- The only feedback you have is your validation metrics.
- Random classifier:
  a. MNIST digit-classification example, a validation accuracy around 0.1.
  b. IMDB review-classification example, a validation accuracy around 0.5.
- Weather forecast example.
- Try a **simple ML** model as linear regression for regression problems, or logistic regression for classification problems.
- **Human performance** on a particular task.

## Things to keep in mind about model evaluation

- Data representativeness: both training and test sets must represent all the classes.
- The arrow of time in time series: data should not be randomly shuffled before splitting it. Avoid **temporal leaks**.
- Redundancy in your data: make sure your training set and validation set are **disjoint**.

## 3.
## Improving model fit

## Receipt for great modeling

1. The initial goal is to achieve a model that shows some **generalization** power and that is able to **overfit**.
2. Once you have such a model, you'll focus on refining generalization by fighting overfitting.

To achieve the perfect fit, you must first overfit!

## Three common problems

1. Training doesn’t get started.
2. Training gets started, but the model doesn’t generalize. It can’t beat the common-sense baseline.
3. Training and validation loss both go down over time, and it can beat your baseline, but it  doesn’t overfit, still underfit.

## 1. Tuning key gradient descent parameters

Training doesn’t get started.

- Sometimes training doesn’t get started, or it stalls too early.
- Remember that you can fit a model to random data.
- Always a problem with the configuration of the gradient descent process:
  - your choice of optimizer,
  - the distribution of initial values in the weights of your model,
  - your learning rate, or
  - your batch size.

**Continue on the notebook...**

## 1. Tuning key gradient descent parameters

If you find yourself in a similar situation, try the following:

- Lowering or increasing the **learning rate**:
  - A learning rate that is **too high** may lead to updates that vastly overshoot a proper fit, like in the preceding example,
  - And a learning rate that is **too low** may make training so slow that it appears to stall.
- **Increasing the batch size**: A batch with more samples will lead to gradients that are more informative and less noisy (lower variance).

## 2. Leveraging better architecture priors

Training gets started, but the model doesn’t generalize.

- Validation metrics aren’t improving at all for some reason.
- It indicates that something is fundamentally wrong with your approach, and it may not be easy to tell what.
- Here are some tips:
  1. The input data imply doesn’t contain sufficient information to predict your targets.
  2. Leverage the right architecture priors: use a model that makes the right assumptions about the problem is essential to achieve generalization.

## 3. Increasing model capacity

Training and validation loss both go down over time, and it can beat your baseline, but it doesn’t overfit, still underfit.

- You need to get your model to start overfitting
- A problem with the representational power of your model: a bigger model is needed.

4.

## Improving generalization

Switch your focus to maximizing generalization

## Dataset curation

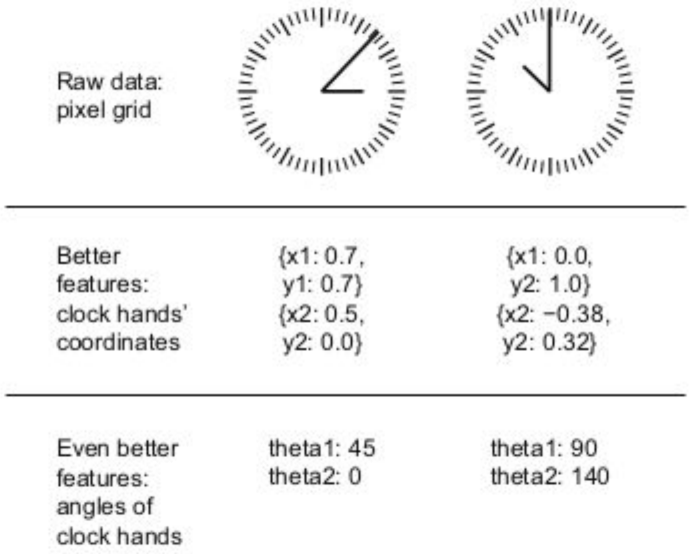
Spending more effort and money on data collection almost always yields a much greater return on investment than spending the same on developing a better model.

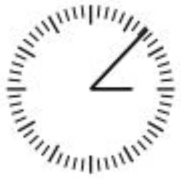
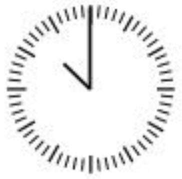
1. Make sure you have enough data:  you need a **dense sampling** of the input-cross-output space.
2. Minimize labeling errors.
3. Clean your data and deal with missing values and duplicate values.
4. Do feature selection If you have many features and you aren’t sure which ones are actually useful.
5.

## Feature engineering

FE is to make the  algorithm work better by applying hardcoded (non-learned) transformations to the data before it goes into the model.

- The data needs to be presented to the model in a way that will make the model’s job easier.
- Good features still allow you to solve problems more elegantly while using fewer resources.
- Good features let you solve a problem with far less data.



| | | |
|---|---|---|
| Raw data: pixel grid |  |  |
| Better features: clock hands' coordinates | {x1: 0.7, y1: 0.7} {x2: 0.5, y2: 0.0} | {x1: 0.0, y2: 1.0} {x2: −0.38, y2: 0.32} |
| Even better features: angles of clock hands | theta1: 45 theta2: 0 | theta1: 90 theta2: 140 |

## Using early stopping

Finding the exact point between an underfit curve and an overfit curve is one of the most effective things you can do to improve generalization.

- Start by training our models for longer epochs than needed to figure out the number of epochs that yielded the best validation metrics.
- Then retrain a new model for exactly that number of epochs.
- Or interrupt training as soon as validation metrics have stopped improving with `callback_early_stopping()` in Keras.

## Regularizing your model

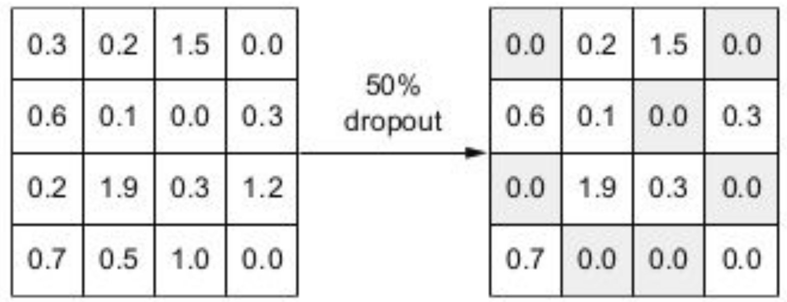
Regularization techniques are a set of best practices to make the model less specific to the training set and better able to generalize.

More common regularization techniques:

- **Reducing the network size**: If the model has limited memorization resources, it won't be able to simply memorize its training data.
- **L1 regularization**: The cost added is proportional to the absolute value of the weight coefficients
- **L2 regularization**: The cost added is proportional to the square of the value of the
- weight coefficients

## Dropout

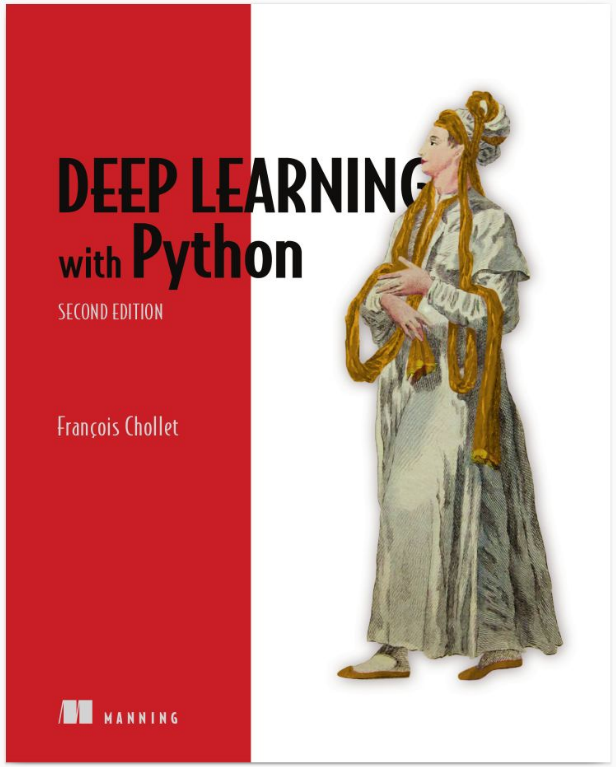
- Dropout is one of the most effective and most commonly used regularization techniques for neural networks.
- Dropout, applied to a layer, consists of randomly setting to zero a number of output features of the layer during training
- It’s usually set between 0.2 and 0.5.
- At test time, no units are dropped out.



## 5.

## Book

## Deep Learning with Python, 2nd Ed. by Francois Chollet



- Chapter 5

# Thanks!

## Any questions?

You can find me at:
rodralez@gmail.com