DATA SCIENCE IN (ASTRO)PARTICLE PHYSICS and COSMOLOGY: the BRIDGE to INDUSTRY

Machine learning: an introduction

Márcio Ferreira

Centro de Física da Universidade de Coimbra

27 - 30 June (2022)

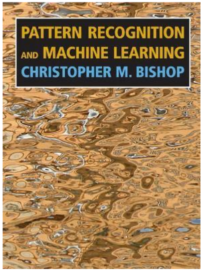
Contact: marcio.ferreira@uc.pt

Bibliography

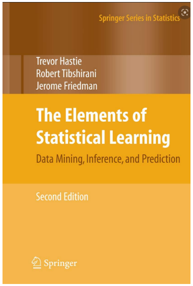
Physics Reports
Volume 810, 30 May 2019, Pages 1-124

A high-bias, low-variance introduction to Machine Learning for physicists

Pankaj Mehta [a], Marin Bukov [b], Ching-Hao Wang [a], Alexandre G.R. Day [a], Clint Richardson [a], Charles K. Fisher [c], David J. Schwab [d]



PATTERN RECOGNITION AND MACHINE LEARNING
CHRISTOPHER M. BISHOP



Springer Series in Statistics

Trevor Hastie
Robert Tibshirani
Jerome Friedman

The Elements of Statistical Learning
Data Mining, Inference, and Prediction

Second Edition

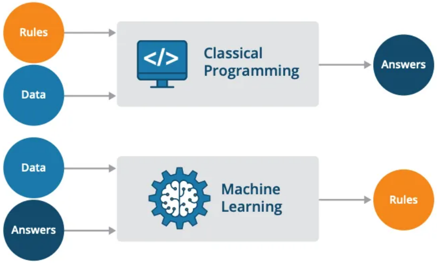
Springer

- *A high-bias, low-variance introduction to machine learning for physicists.*, Mehta, Pankaj, et al., Physics reports 810 (2019)
- *Pattern recognition and machine learning*, Bishop, Christopher M., springer (2006)
- *The elements of statistical learning: data mining, inference, and prediction*, Hastie, Trevor, Robert Tibshirani, and Jerome Friedman, Springer Science Business Media (2009)

Contents

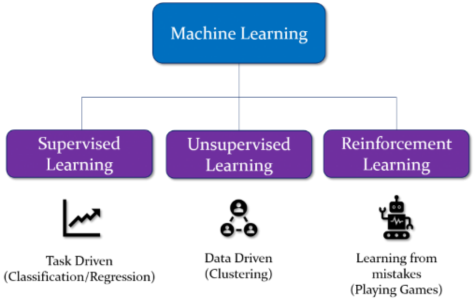
AI and Machine learning

- Artificial intelligence (AI) (1950s) comprises a great variety of sub-fields from science to engineering.
    - Understand the basis of human intelligence and replicate it on intelligent entities.
- Machine learning (ML) is a sub-field of AI:
    - **Are computers able to perform a specific task by automatically learn the required rules from data?**
- Instead of being explicitly programmed, ML systems are trained.



Machine learning: different types

- **Supervised learning:** learns from label data
- **Unsupervised learning:** finds patterns in unlabeled data
- **Reinforcement learning:** learning system interacts with the environment and takes suitable actions to maximize reward in a particular situation.

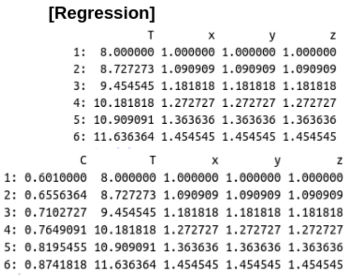


Supervised machine learning: dataset

- We need data

$$\mathcal{D} = [(\mathbf{y}_1, \mathbf{x}_1), (\mathbf{y}_2, \mathbf{x}_2), \cdots, (\mathbf{y}_N, \mathbf{x}_N)]$$

where $\mathbf{y}/\mathbf{x}$ are the dependent/independent variables

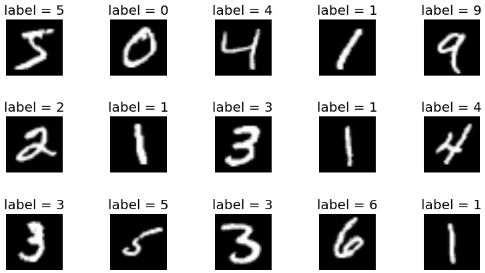


[Regression]

```
                T         x         y         z
    1:  8.000000 1.000000 1.000000 1.000000
    2:  8.727273 1.090909 1.090909 1.090909
    3:  9.454545 1.181818 1.181818 1.181818
    4: 10.181818 1.272727 1.272727 1.272727
    5: 10.909091 1.363636 1.363636 1.363636
    6: 11.636364 1.454545 1.454545 1.454545
```

```
           C         T         x         y         z
1: 0.6010000  8.000000 1.000000 1.000000 1.000000
2: 0.6556364  8.727273 1.090909 1.090909 1.090909
3: 0.7102727  9.454545 1.181818 1.181818 1.181818
4: 0.7649091 10.181818 1.272727 1.272727 1.272727
5: 0.8195455 10.909091 1.363636 1.363636 1.363636
6: 0.8741818 11.636364 1.454545 1.454545 1.454545
```

Handwritten digits (MNIST) [classification]



Supervised machine learning: model

- A model $\mathbf{y} = f(\mathbf{w}, \mathbf{x}) \equiv f_{\mathbf{w}}(\mathbf{x})$ is a map between elements in the input and output spaces

$$f_{\mathbf{w}} : \mathbf{x} \longrightarrow \mathbf{y}$$

where $\mathbf{y}/\mathbf{x}$ are the dependent/independent variables

- We have a model once $\mathbf{w}$ is fixed (the map is defined)
- Linear models (linear in $\mathbf{w}$)

$$f(\mathbf{w}, \mathbf{x}) = b + w_1 x_1 + ... + w_n x_n = \mathbf{w} \cdot \mathbf{x}$$

- Class of linear models [model complexity]

$$y = f_1(\mathbf{w}, \mathbf{x}) = w_1 x + b \qquad \text{[all polynomials of order 1]}$$
$$y = f_2(\mathbf{w}, \mathbf{x}) = w_1 x + w_2 x^2 + b \qquad \text{[all polynomials of order 2]}$$
$$y = f_5(\mathbf{w}, \mathbf{x}) = w_1 x + w_2 x^2 + ... + w_5 x^5 + b \qquad \text{[all polynomials of order 5]}$$

Supervised machine learning: model's performance

- **Cost function** measures the deviation between the model's predictions and the *true* values

$$\mathcal{C}(\mathbf{y}, \hat{\mathbf{y}}(\mathbf{x}; \mathbf{w}))$$

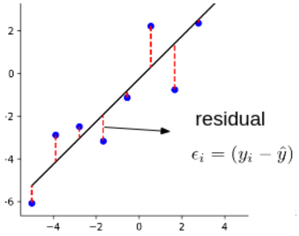
- Regression problems: **mean squared error (MSE)**

$$\mathcal{C}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}(\mathbf{w}, \mathbf{x}_i))^2 \quad \text{(training dataset)}$$

- **Loss function** (squared error):

$$l_i = (y_i - \hat{y}(\mathbf{w}, \mathbf{x}_i))^2 = \epsilon_i^2 \quad \text{(single point)}$$

$$\mathcal{C}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N}\sum_{i=1}^{N} l_i = \frac{1}{N}\sum_{i=1}^{N} \epsilon_i^2$$



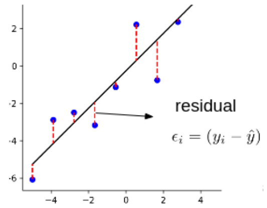
Supervised machine learning: training

- **Training a model**: finding the values $\mathbf{w}_*$ that minimizes the cost function

$$\mathbf{w}_* = \arg\min_{\mathbf{w}} \ \mathcal{C}(y, \hat{y}_\lambda(\mathbf{w}, \mathbf{x}))$$

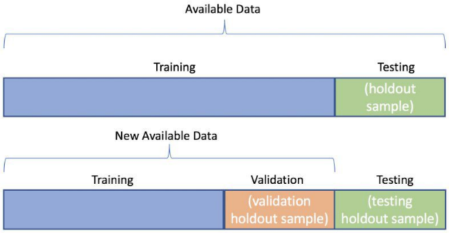
- We are minimizing the mean residuals

$$\mathbf{w}_* = \arg\min_{\mathbf{w}} \ \frac{1}{N} \sum_{i}^{N} (\epsilon_i(\mathbf{w}))^2$$



Finding predicting models: data splitting

- **Randomly** split your dataset:
  - Train set: $\mathcal{D}_{train}$ (%80 of $\mathcal{D}$)
  - Test set: $\mathcal{D}_{test}$ (%20 of $\mathcal{D}$)
- **Randomly** split $\mathcal{D}_{train}$:
  - Train set: $\mathcal{D}_{train}$ (%80)
  - Validation set: $\mathcal{D}_{val}$ (%20)



- **Train set**: train your candidate models
- **Validate set**: select the best model among the candidate models
- **Testing set**: evaluate the *real* accuracy of the best model

Several candidate models: finding the best model

- Train the different models $\hat{y}_1$, $\hat{y}_2$, $\hat{y}_3$, ..., $\hat{y}_k$ in $\mathcal{D}_{train}$

$$\mathcal{C}(y, \hat{y}) = \frac{1}{J} \sum_{i=1}^{J} (y_i^{train} - \hat{y}_{\lambda}(\mathbf{w}^{\lambda}, \mathbf{x}_i^{train}))^2$$

- Find the best parameters $\mathbf{w}_*^{\lambda}$ for each model λ

$$\mathbf{w}_*^{\lambda} = \arg\min_{\mathbf{w}^{\lambda}} \; \mathcal{C}(y, \hat{y}_{\lambda}(\mathbf{w}^{\lambda}, \mathbf{x}))$$

- **Select the best model** $\hat{y}_{\lambda}$ **in** $\mathcal{D}_{val}$

$$\mathcal{C}(y, \hat{y}) = \frac{1}{L} \sum_{i=1}^{L} (y_i^{val} - \hat{y}_{\lambda}(\mathbf{w}_*^{\lambda}, \mathbf{x}_i^{val}))^2$$

Performance of the final model

- Determine the final performance of the best model in the $\mathcal{D}_{test}$

$$\mathcal{C}(y, \hat{y}) = \frac{1}{T}\sum_{i=1}^{T}(y_i^{test} - \hat{y}_{model}(\mathbf{w}_*, \mathbf{x}_i^{test}))^2$$

- **Unseen data:** neither used in the training nor in validating stages

In-sample and out-of-sample errors

- The model **in-sample error** is

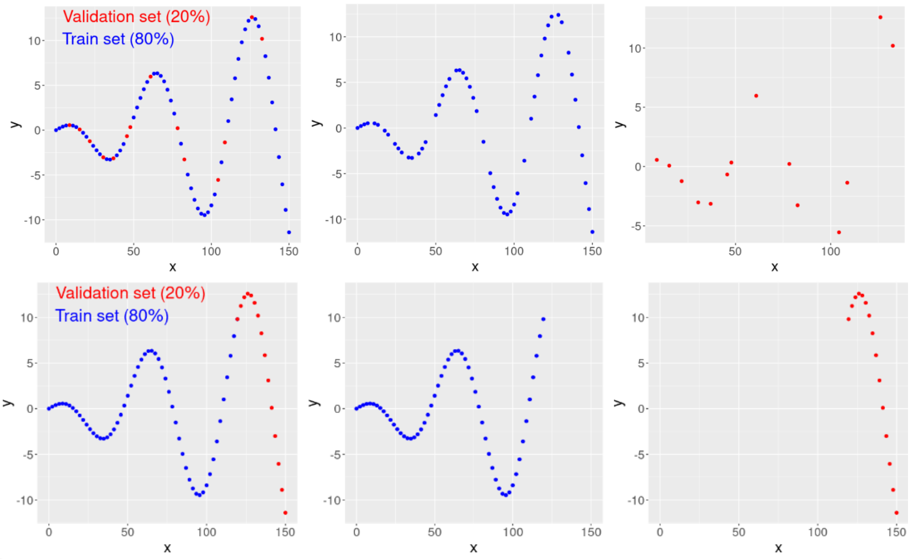
$$E_{\mathsf{in}} = \mathcal{C}(\mathbf{y}_{train}, \hat{\mathbf{y}}(\hat{\mathbf{w}}, \mathbf{x}_{train})) = \frac{1}{K}\sum_{i=1}^{J}(y_i^{train} - \hat{y}(\hat{\mathbf{w}}, \mathbf{x}_i^{train}))^2$$

- The model **out-of-sample error** is

$$E_{\mathsf{out}} = \mathcal{C}(\mathbf{y}_{val}, \hat{\mathbf{y}}(\hat{\mathbf{w}}, \mathbf{x}_{val})) = \frac{1}{M}\sum_{i=1}^{M}(y_i^{val} - \hat{y}(\hat{\mathbf{w}}, \mathbf{x}_i^{val}))^2$$

- In general $E_{\mathsf{out}} \geq E_{\mathsf{in}}$
- The random split of $\mathcal{D}$ into $\mathcal{D}_{train}$ and $\mathcal{D}_{val}$ ensures an unbiased estimate of the model's performance **(cross-validation)**
- We select the model with lowest E_{out}

Why do we need random splitting?



Linear regression

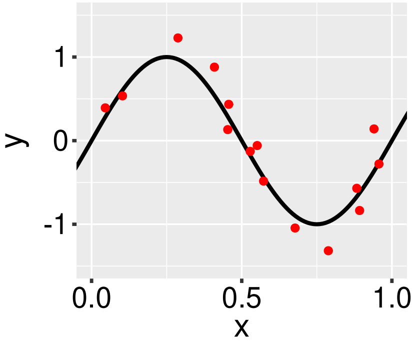
- Assume the following generating process (synthetic dataset)

$$y(x) = \sin(2\pi x) + \eta$$

- **Gaussian noise** simulates real data:

$$\eta \sim \mathcal{N}(\mathbf{0}, \mathbf{0.2}) \longrightarrow \langle \eta \rangle = 0 \text{ and } \langle \eta_i \eta_j \rangle = 0.2\delta_{ij}$$

- $N = 15$ *"observations"* uniformly spaced in $[0, 1]$



Linear regression

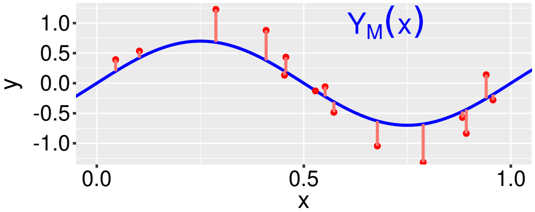
- **Goal of ML**: learning the underlying process $[\sin(2\pi x)]$ from $\mathcal{D}_{train}$
- Select a class model (polynomial function of order M):

$$\hat{\mathbf{y}}_{\mathbf{M}}(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + ... + w_M x^M$$

  - Linear model: $y_M(x, \mathbf{w})$ is linear in $\mathbf{w}$

- Define a cost function:

$$\mathcal{C}(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} \left[\hat{\mathbf{y}}_{\mathbf{M}}(\mathbf{x}, \mathbf{w}) - y_n\right]^2 \qquad \text{(MSE)}$$



Linear regression: learning process

- **Learning processes**: determining $\mathbf{w}$ that minimizes $\mathcal{C}(\mathbf{w})$
- **Closed-form solution:** derivatives $\frac{\partial \mathcal{C}}{\partial \mathbf{w}}$ are linear in $\mathbf{w}$

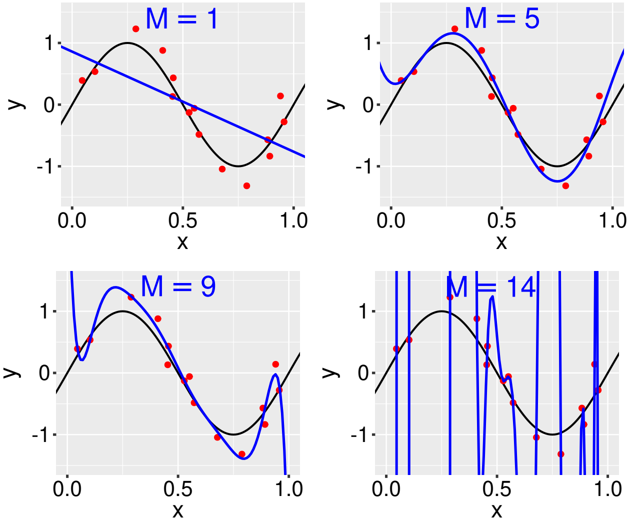
$$\mathbf{w}^* = \left(X^T X\right)^{-1} X^T y \quad \text{(ordinary least squares)}$$

- Our fitted models still depends on M:

$$y_M(x, \mathbf{w}^*) \quad \text{with} \quad \mathbf{w}^* \equiv \mathbf{w}^*(M)$$

- **Model selection:** choose the polynomial's order M that best fits data

Linear regression: overfitting/underfitting



- **Underfitting:** $y_1(x)$ is a poor representation
- **Overfitting:** $y_{14}(x)$ is a poor representation ($E_{in} = 0$). **Why?**

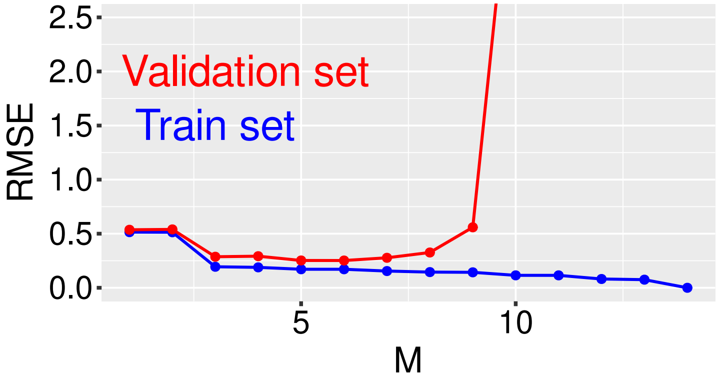
Linear regression: model's generalization

- **However, we are interested in E_{out}**
  - a measure of the generalization capacity of the model, i.e., making accurate predictions for unseen data
- How does E_{out} depend on M?
  - We generate a validation set with $K = 120$ points
  - We use the root-mean-square error (RMSE)

$$E_{\text{RMSE}} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{K}\sum_{n=1}^{K}(\hat{y}_n - y_n)^2} \quad \text{(models' accuracy)}$$

  - E_{RMSE} has the same units and scale as the target variable y

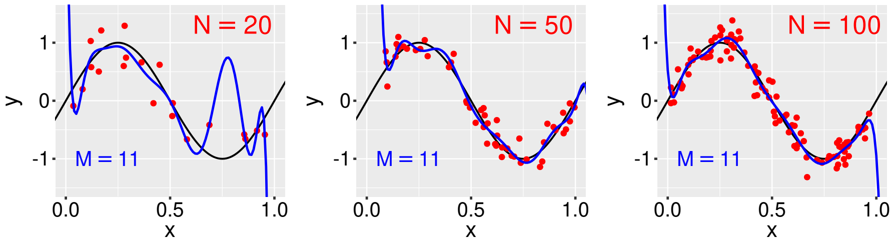
Linear regression: cross-validation



- What is the best model and why?

Linear regression problem: dataset's size

- **Effect of the dataset size**



- Over-fitting becomes less severe as N grows (fixed model complexity)
- However, the model's complexity should be chosen according to the complexity of the problem and not the data set size

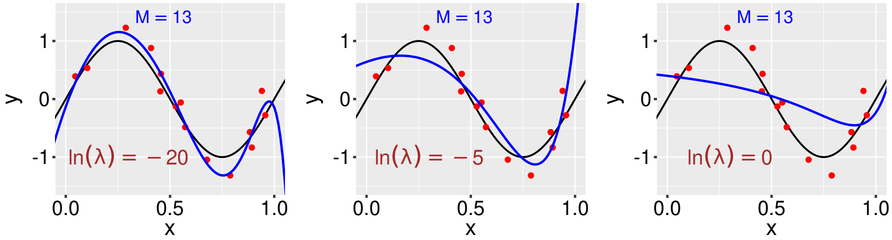
Regularization technique I

- Regularization controls the overfitting phenomena by adding a penalty factor in the cost function

$$\tilde{C}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{n}[y_M(x,\mathbf{w}) - y_n]^2 + \frac{\lambda}{2}||\mathbf{w}||^2$$

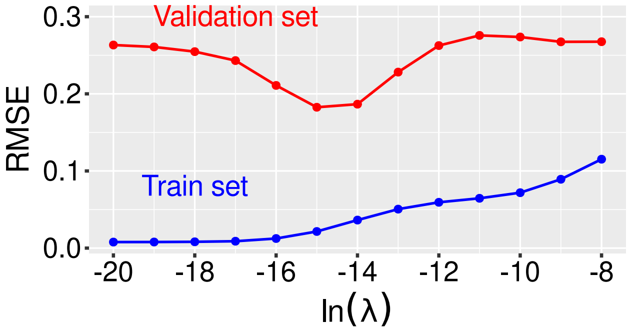
with $||\mathbf{w}||^2 = \mathbf{w}^T\mathbf{w} = w_0^2 + w_1^2 + ... + w_N^2$

- λ controls the amount of penalty [OLS solution for $\lambda = 0$]
- Required when complex models are applied to small datasets



Regularization technique II

- λ **controls the effective complexity of the models** and the degree of over-fitting
- **Use cross-validation to select the best λ**



- What is the best model (value of λ) and why?

Common regression regularizations

- Ridge regression (L_2):

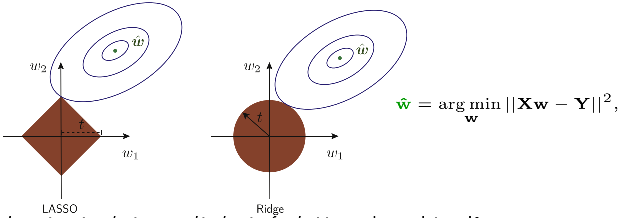
$$\hat{\mathbf{w}}_{\mathbf{R}} = \arg\min_{\mathbf{w}} \left\{||\mathbf{X}\mathbf{w} - \mathbf{Y}||^2 + \lambda||\mathbf{w}||^{\mathbf{2}}\right\}$$

$$\hat{\mathbf{w}}_{\mathbf{R}} = \arg\min_{\mathbf{w}} ||\mathbf{X}\mathbf{w} - \mathbf{Y}||^2, \text{ subject to: } ||\mathbf{w}||^{\mathbf{2}} \leq t$$

- LASSO regression (L_1):

$$\hat{\mathbf{w}}_{\mathbf{L}} = \arg\min_{\mathbf{w}} \left\{||\mathbf{X}\mathbf{w} - \mathbf{Y}||^2 + \lambda||\mathbf{w}||^{\mathbf{1}}\right\}$$

$$\hat{\mathbf{w}}_{\mathbf{L}} = \arg\min_{\mathbf{w}} ||\mathbf{X}\mathbf{w} - \mathbf{Y}||^2, \text{ subject to: } ||\mathbf{w}||^{\mathbf{1}} \leq t$$

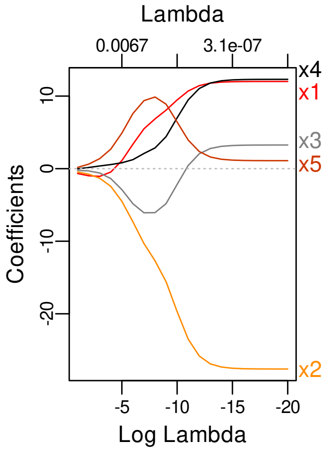

$\hat{w}$
w_2
t
w_1
LASSO
$\hat{w}$
w_2
t
w_1
Ridge
$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} ||\mathbf{X}\mathbf{w} - \mathbf{Y}||^2,$

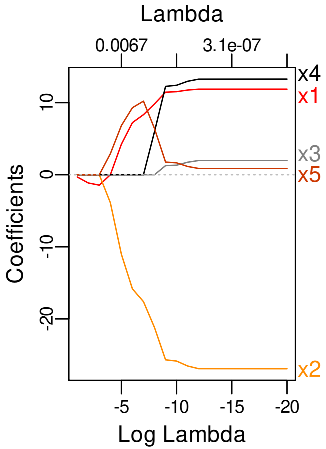
[***A high-bias, low-variance introduction to machine learning for physicists.***, **Mehta, Pankaj, et al.**]

- LASSO tends to give sparse solutions

Regression regularizations: Ridge vs LASSO

- Model: $y_5(x)$





Statistical learning theory: Bias-Variance decomposition

- We run an experiment and collect a dataset $\mathcal{D}_\mathbf{i} = (\mathbf{x}, y)$
  - The system's dynamics is governed/generated by $y = f(\mathbf{x}) + \eta$
- Our model is given by $f(\mathbf{x}, \hat{\mathbf{w}}_{\mathbf{D_i}})$

$$\hat{\mathbf{w}}_{\mathbf{D_i}} = \arg\min_{\mathbf{w}} \mathcal{C}(\mathbf{w}) = \arg\min_{\mathbf{w}} \sum_{n=1}^{M} [f(\mathbf{x}_n, \mathbf{w}) - y_n]^2$$

- **$\hat{\mathbf{w}}_{\mathbf{D_i}}$ is a function of the dataset $\mathcal{D}_\mathbf{i}$**

- Performing N times the experiment (M samples): $\mathcal{D}_\mathbf{1}, \mathcal{D}_\mathbf{2}, ..., \mathcal{D}_\mathbf{N}$
- We obtain N models: $\hat{\mathbf{w}}_{\mathbf{D_1}}, \hat{\mathbf{w}}_{\mathbf{D_2}}, ..., \hat{\mathbf{w}}_{\mathbf{D_N}}$

Statistical learning theory: Bias-Variance decomposition

- An unbiased estimate of a model's uncertainty must consider all possible datasets $\hat{\mathbf{w}}_{\mathbf{D_i}}$ and realizations of the noise η

- The out-of-sample (generalization) error is

$$
\begin{aligned}
E_{\text{out}} &= \mathbb{E}_{\mathbf{D},\eta}\left[\mathcal{C}(y, f(\mathbf{x}, \hat{\mathbf{w}}_D))\right] \\
&= \mathbb{E}_{\mathbf{D},\eta}\left[\sum_{n=1}^{N}\left[y_n - f(\mathbf{x}_n, \hat{\mathbf{w}}_D)\right]^2\right] \\
&= \text{Bias}^2 + \text{Variance} + \text{Noise}
\end{aligned}
$$

Statistical learning theory: Bias-Variance decomposition

$$\text{Bias} = \sum_n \left(y(x_n) - \mathbb{E}_{\mathcal{D}}\left[f(\mathbf{x_n}, \hat{\mathbf{w}}_D)\right]\right)$$

deviation of the model's asymptotic prediction from the true value

$$\text{Variance} = \sum_n \mathbb{E}_{\mathcal{D}}\left[\left(f(\mathbf{x_n}, \hat{\mathbf{w}}_{\mathbf{D}}) - \mathbb{E}_{\mathcal{D}}\left[f(\mathbf{x_n}, \hat{\mathbf{w}}_{\mathbf{D}})\right]\right)^2\right]$$

how much our model fluctuates around its mean (finite-sample effects)

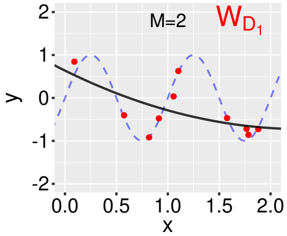
$$\text{Noise} = \sum_n \sigma_\eta^2$$

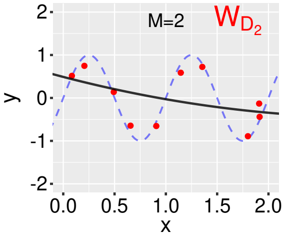
irreducible error (lower bound on E_{out})

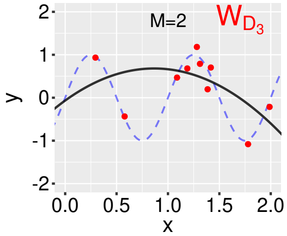
- As the model's complexity increases, it captures more complex patterns decresing the bias.
- On the other hand, the model's predictions strongly fluctuates as its complexity increases when trained in different sets.

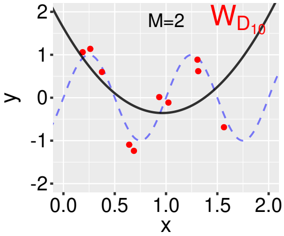
Bias-Variance decomposition

- Model: $\hat{y}_M(x, \mathbf{w}) = w_0 + w_1 x + \cdots + w_M x^M \quad [y(x) = \sin(2\pi x) + \eta]$
- *"Repeating the experiment"* 10 times: $\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_{10}$
- We obtain 10 models: $\hat{\mathbf{w}}_{D_1}, \hat{\mathbf{w}}_{D_2}, ..., \hat{\mathbf{w}}_{D_{10}}$



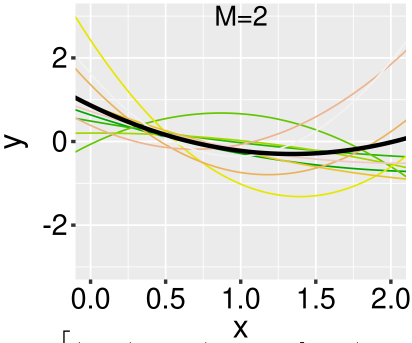


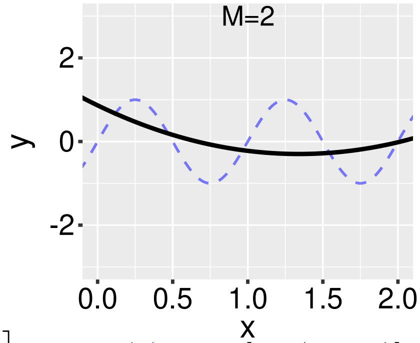




Bias-Variance decomposition

- Color lines: The fitted models $\hat{y}_M(\hat{\mathbf{w}}_{D_1}), \hat{y}_M(\hat{\mathbf{w}}_{D_2}), \cdots, \hat{y}_M(\hat{\mathbf{w}}_{D_{10}})$
- **Solid line:** $\mathbb{E}_{\mathcal{D}}\left[\hat{y}_M(x, \hat{w}_D)\right]$ (mean value of our estimator)
- Dashed line: $y(x) = \sin(2\pi x)$

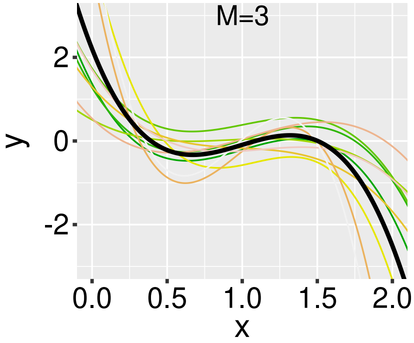


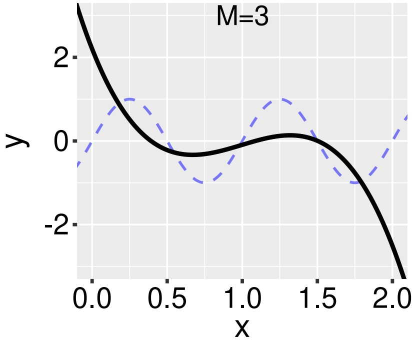


$$\underbrace{\mathbb{E}_{\mathcal{D}}\left[\left(\hat{y}_M(x, \hat{w}_D) - \mathbf{E}_{\mathcal{D}}\left[\hat{\mathbf{y}}_{\mathbf{M}}(\mathbf{x}, \hat{\mathbf{w}}_{\mathbf{D}})\right]\right)^2\right]}_{\textbf{Variance}}$$

$$\underbrace{y(x) - \mathbf{E}_{\mathcal{D}}\left[\hat{\mathbf{y}}_{\mathbf{M}}(\mathbf{x}, \hat{\mathbf{w}}_{\mathbf{D}})\right]}_{\textbf{Bias}}$$

Bias-Variance decomposition

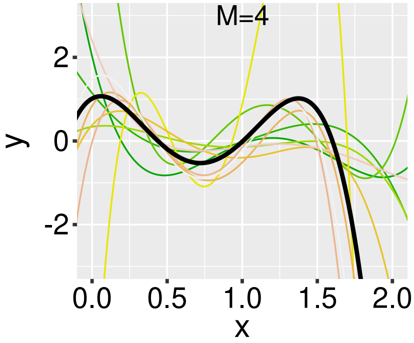


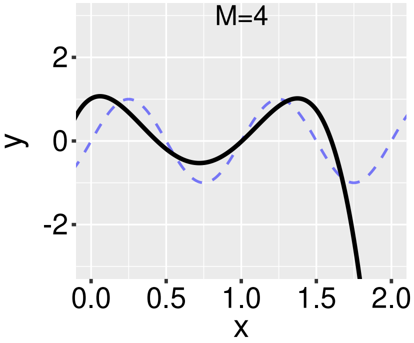


$$\underbrace{\mathbb{E}_{\mathcal{D}}\left[\left(\hat{y}_M(x,\hat{w}_D) - \mathbf{E}_{\mathcal{D}}\left[\hat{\mathbf{y}}_{\mathbf{M}}(\mathbf{x},\hat{\mathbf{w}}_{\mathbf{D}})\right]\right)^2\right]}_{\textbf{Variance}}$$

$$\underbrace{{\color{blue}y(x)} - \mathbf{E}_{\mathcal{D}}\left[\hat{\mathbf{y}}_{\mathbf{M}}(\mathbf{x},\hat{\mathbf{w}}_{\mathbf{D}})\right]}_{\textbf{Bias}}$$

Bias-Variance decomposition

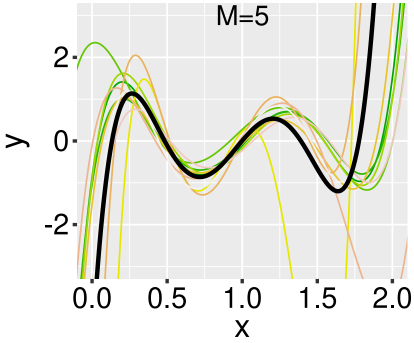


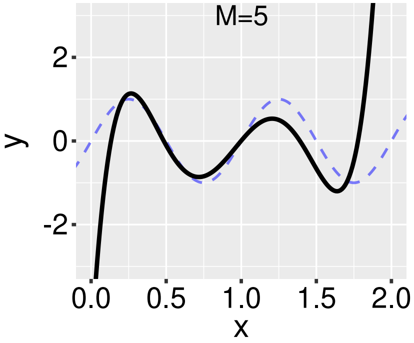


$$\underbrace{\mathbb{E}_{\mathcal{D}}\left[\left(\hat{y}_M(x,\hat{w}_D) - \mathbf{E}_{\mathcal{D}}\left[\hat{\mathbf{y}}_{\mathbf{M}}(\mathbf{x},\hat{\mathbf{w}}_{\mathbf{D}})\right]\right)^2\right]}_{\textbf{Variance}}$$

$$\underbrace{\color{blue}{y(x)}\color{black} - \mathbf{E}_{\mathcal{D}}\left[\hat{\mathbf{y}}_{\mathbf{M}}(\mathbf{x},\hat{\mathbf{w}}_{\mathbf{D}})\right]}_{\textbf{Bias}}$$

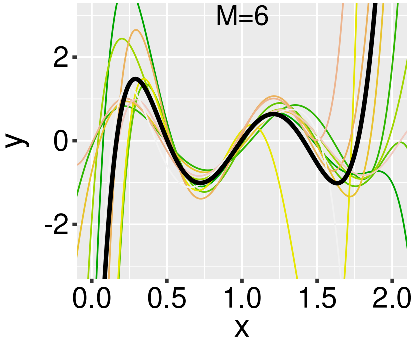
Bias-Variance decomposition

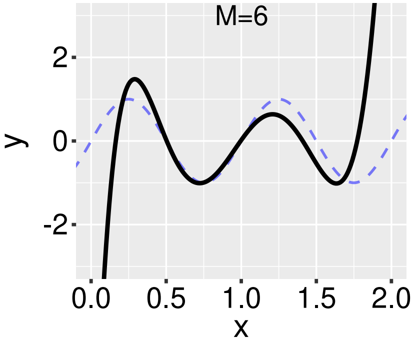




$$\underbrace{\mathbb{E}_{\mathcal{D}}\left[\left(\hat{y}_M(x,\hat{w}_D) - \mathbf{E}_{\mathcal{D}}\left[\hat{\mathbf{y}}_{\mathbf{M}}(\mathbf{x},\hat{\mathbf{w}}_{\mathbf{D}})\right]\right)^2\right]}_{\textbf{Variance}} \qquad \underbrace{y(x) - \mathbf{E}_{\mathcal{D}}\left[\hat{\mathbf{y}}_{\mathbf{M}}(\mathbf{x},\hat{\mathbf{w}}_{\mathbf{D}})\right]}_{\textbf{Bias}}$$

Bias-Variance decomposition



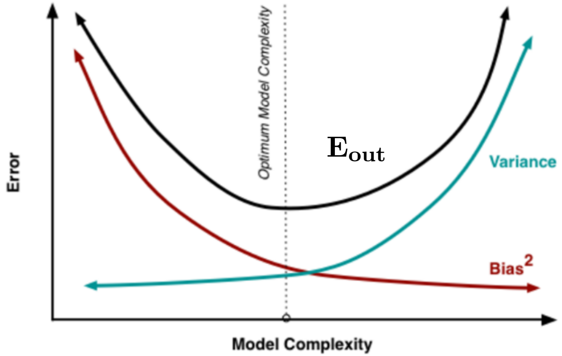


$$\underbrace{\mathbb{E}_{\mathcal{D}}\left[\left(\hat{y}_M(x,\hat{w}_D) - \mathbf{E}_{\mathcal{D}}\left[\hat{\mathbf{y}}_{\mathbf{M}}(\mathbf{x},\hat{\mathbf{w}}_{\mathbf{D}})\right]\right)^2\right]}_{\textbf{Variance}}$$

$$\underbrace{{\color{blue}y(x)} - \mathbf{E}_{\mathcal{D}}\left[\hat{\mathbf{y}}_{\mathbf{M}}(\mathbf{x},\hat{\mathbf{w}}_{\mathbf{D}})\right]}_{\textbf{Bias}}$$

Bias-Variance decomposition

- The role of the model's complexity for finite amount of data



Training a ML model: convex problem

- Optimization problem for linear models $[\hat{y}(\mathbf{w}, \mathbf{x}) = \mathbf{w}.\mathbf{x}]$

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \mathcal{C}(\mathbf{y}, \hat{\mathbf{y}}) = \arg\min_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}(\mathbf{w}, \mathbf{x}_i))^2$$

- As $\mathcal{C}$ is quadratic in $\mathbf{w}$ with positive-definite Hessian

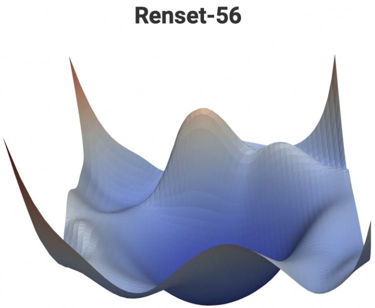
$$\mathbf{w}^T H \mathbf{w} > 0, \text{ where } H_{ij} = \nabla_{w_i} \nabla_{w_j} \mathcal{C}$$

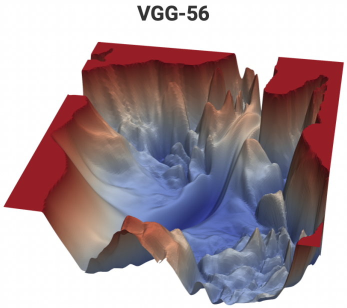
- we have a convex problem: $\mathcal{C}$ has a global minimum at $\mathbf{w}^*$
- OLS solution $(\nabla_{\mathbf{w}} \mathcal{C} = 0)$

$$\mathbf{w}^* = \left(X^T X\right)^{-1} X^T y$$

Training a ML model: non-convex problem

- However, **ML models** usually have complex **non-convex cost functions in a high-dimensional space with many local minima**.





[https://www.cs.umd.edu/ tomg/projects/landscapes/]

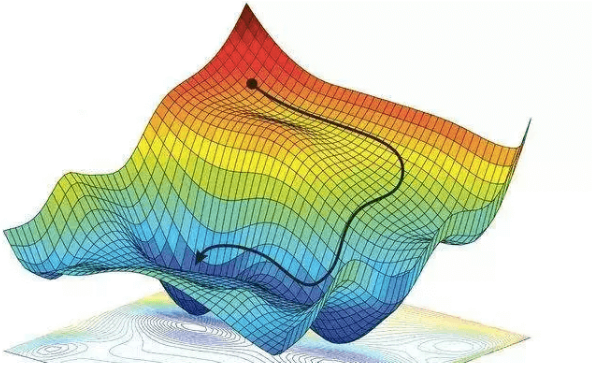
- There is no closed-form solution and gradient descent algorithms are used to numerically search the solution.

Gradient descent methods

- **Gradient descent method:** solves the optimization problem

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \mathcal{C}(\mathbf{x}, \mathbf{y}, \mathbf{w})$$

by adjusting $\mathbf{w}$, in successive iterations, in the direction where $\nabla\mathcal{C}(\mathbf{x}, \mathbf{y}, \mathbf{w})$ is large and negative (steepest descent)

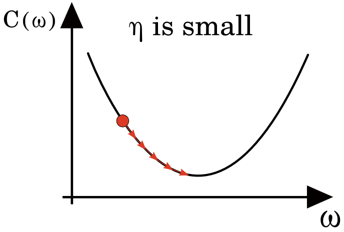


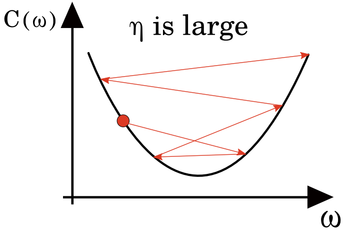
Gradient descent algorithm

- Initialize $\mathbf{w}$ to some value $\mathbf{w}_0$ and update its value according to

$$\mathbf{v}_t = \eta_{\mathbf{t}} \nabla_{\mathbf{w}} \mathcal{C}(\mathbf{w}_t)$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{v}_t$$

- The learning rate $\eta_{\mathbf{t}}$ (step size) is a sensitive parameter:
  - Too many steps for low η_t values
  - It may oscillate and diverge for high η_t values





Gradient descent algorithm

- **GD is a deterministic algorithm**

$$\mathcal{C}(\mathbf{w}) = \sum_{i=1}^{N} \mathcal{C}(\mathbf{y_i}, \mathbf{x_i}, \mathbf{w})$$

- The surface $\mathcal{C}(\mathbf{w})$ **is fixed for a given dataset.**

$$\{(\mathbf{x_1}, \mathbf{y_1}), ..., (\mathbf{x_N}, \mathbf{y_N})\}$$

- **Drawbacks** of GD:
  - Converges to local minimum (when it converges)
  - Sensitive to the value of $\mathbf{w}_0$
  - Sensitive to η_t
  - All directions of $\mathbf{w}$ space are equally treated
  - Computationally expensive for large datasets

Newton's method

How can GD algorithm be improved?

- Make $\eta_{\mathbf{t}}$ sensitive to local surface properties of $\mathcal{C}(\mathbf{w_i})$
- Newton's method (second-order Taylor expansion)

$$\mathcal{C}(\mathbf{w}+\mathbf{v}) \approx \mathcal{C}(\mathbf{w}) + \nabla_{\mathbf{w}}\mathcal{C}(\mathbf{w})\mathbf{v} + \frac{1}{2}\mathbf{v}^T H(\mathbf{w})\mathbf{v}$$

- Update rules

$$\mathbf{v}_t = H^{-1}(\mathbf{w}_t)\nabla_{\mathbf{w}}\mathcal{C}(\mathbf{w}_t)$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{v}_t$$

- The learning rate $\eta_{\mathbf{t}}(\mathbf{w_t}) = \mathbf{H^{-1}}(\mathbf{w_t})$ adjusts the different parameters' step sizes depending on the Hessian matrix:
  - Larger steps in flat directions (small curvature)
  - Smaller steps in steep directions (large curvature)
- **However, the Hessian is expensive to compute.**

Stochastic Gradient descent algorithm

- **Stochastic Gradient descent algorithm:** adding stochasticity
- The full gradients are approximated on a **mini-batch** (subset of data)
- A step in SGD is determined on a single MB as

$$\nabla_{\mathbf{w}} \mathcal{C}^{\mathsf{MB}}(\mathbf{w}) = \sum_{i \in B_k} \nabla_{\mathbf{w}} \mathcal{C}(\mathbf{x}_i, \mathbf{w})$$

- The SGD algorithm

$$\mathbf{v}_t = \eta_t \nabla_{\mathbf{w}} \mathcal{C}^{\mathsf{MB}}(\mathbf{w}_t)$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{v}_t$$

- An **epoch** is a full iteration over all MB (all data points)
- The use of MB speeds up calculations and reduces the probability of getting stuck in local minimum or saddle points.

Adding momentum to SGD

- Introducing a momentum term

$$\mathbf{g}_t = \eta_t \nabla_{\mathbf{w}} \mathcal{C}^{\mathsf{MB}}(\mathbf{w}_t)$$
$$\mathbf{v}_t = \gamma \mathbf{v}_{t-1} + \mathbf{g}_t$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{v}_t$$

  where $\mathbf{0} \leq \gamma < \mathbf{1}$ is the moment parameter.

- $\mathbf{v}_t$ is a running average (memory of the moving direction)

$$\mathbf{v}_t = \mathbf{g}_t + \gamma \mathbf{g}_{t-1} + \gamma^2 \mathbf{g}_{t-2} + \gamma^3 \mathbf{g}_{t-3} + \cdots \quad \text{(time scale of } 1/(1-\gamma)\text{)}$$

- Algorithm speeds up in directions with persistent gradients and suppresses oscillations in high-curvature directions
- Accumulated gradients will help to avoid saddle points

RMSprop algorithm

How to have an **adaptive** η_t without the computationally price of calculating the Hessian?

- The RMSprop algorithm tracks the gradient second momentum to normalize η_t

$$\begin{aligned} \mathbf{g}_t &= \nabla_{\mathbf{w}} \mathcal{C}^{\mathsf{MB}}(\mathbf{w}_t) \\ \mathbf{s}_t &= \beta \mathbf{s}_{t-1} + (1-\beta)\mathbf{g}_t^2 \\ \mathbf{w}_{t+1} &= \mathbf{w}_t - \eta_t \frac{\mathbf{g}_t}{\sqrt{\mathbf{s}_t + \epsilon}} \end{aligned}$$

- β controls the scale time of the second momentum ($\beta \approx 0.9$) and $\epsilon \sim 10^{-8}$ is a regularization term
- The learning rate $\eta_t/\sqrt{\mathbf{s}_t + \epsilon}$ decreases in directions where $\mathbf{g}_t$ is **consistently** large

ADAM algorithm

- The ADAM algorithm tracks of **first** and second momentum of g_t

$$
\begin{aligned}
\mathbf{g}_t &= \nabla_{\mathbf{w}} \mathcal{C}^{\mathsf{MB}}(\mathbf{w}_t) \\
\mathbf{m}_t &= \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t \quad \text{[momentum]} \\
\mathbf{s}_t &= \beta_2 \mathbf{s}_{t-1} + (1-\beta_2)\mathbf{g}_t^2 \quad \text{[RMSprop]} \\
\hat{\mathbf{m}}_{\mathbf{t}} &= \frac{\mathbf{m}_t}{1-(\beta_1)^t} \quad \text{[bias correction]} \\
\hat{\mathbf{s}}_t &= \frac{\mathbf{s}_t}{1-(\beta_2)^t} \quad \text{[bias correction]} \\
\mathbf{w}_{t+1} &= \mathbf{w}_t - \eta_t \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{s}}_t}+\epsilon} \quad \text{[RMSprop+momentum]}
\end{aligned}
$$

- The equations $\hat{\mathbf{m}}_{\mathbf{t}}$ and $\hat{\mathbf{m}}_{\mathbf{t}}$ correct the effect of $\mathbf{m}_0 = \mathbf{s}_0 = 0$
- ADAM is a combination of RMSProp and SGD with momentum.

Algorithm comparison

gif1 gif2

- The SGD algorithm is usually sufficient for simple models.
- SGD+momentum and RMSprop are good options to increase the model accuracy.
- For complex ML models, like deep neural networks, the ADAM algorithm is quite popular and the standard one.

Conclusions

- Basic concepts of supervised machine learning
- Split of the dataset: train, validation, and test
- Regularization techniques (L_1 and L_2): control the model's complexity
- Bias-variance composition: tension between bias and variance terms
- Different gradient descent algorithms:
    - Stochasticity (SGD)
    - Momentum (SGD + momentum)
    - Adaptive learning rate (RMSporp and ADAM)

- **Next lecture:** Deep Neural Networks