ECE521 W17 Tutorial 2*

Kaustav Kundu & Shenlong Wang

UNIVERSITY OF TORONTO

*Reference: Jimmy Ba Lecture Slides, Kevin Swersky 411 Optimization tutorial

Schedule

- Optimization
  - Introduction
  - Gradient Descent
  - Momentum, Nesterov Accelerated Momentum
  - Learning Rate Schedulers
- Gaussian Distribution
- Overfitting and Underfitting

Optimization: An Informal Definition

- Minimize (or maximize) some quantity.

Applications

- Engineering: Minimize fuel consumption of an automobile
- Economics: Maximize returns on an investment
- Supply Chain Logistics: Minimize time taken to fulfill an order
- Life: Maximize happiness

Formal definition

- Goal: find $\theta^* = \arg\min_{\theta} f(\theta)$
- Optimization variable: $\theta \in \mathbb{R}^n$
- Objective function $f : \mathbb{R}^n \to \mathbb{R}$
- Maximize f is equivalent to minimize -f

Optimization is a large area of research

The best method varies depending on specific problems:

- Is the optimization variable discrete or continuous?
- Is the objective function well-behaved? (linear, differentiable, convex, submodular, etc.)
- Do we have the constraints on the variable?
- ….

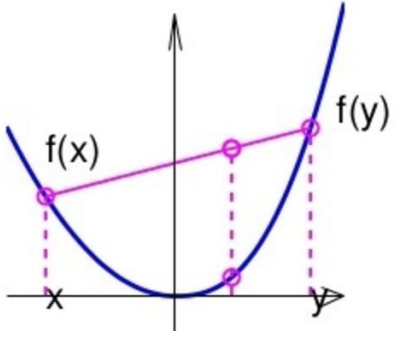
Convex Function

- A function is convex iff for any two points W1 and W2:

$$\forall \alpha \in [0, 1]$$

$$f(\alpha W_1 + (1 - \alpha) W_2) \leq \alpha f(W_1) + (1 - \alpha) f(W_2)$$

convex



non-convex

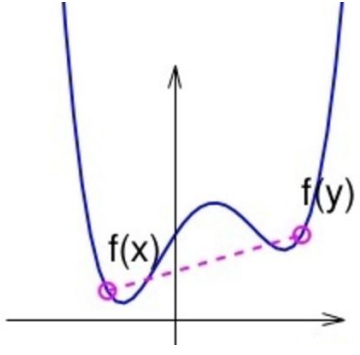


*Image credit: Jimmy Ba

Convex Function

Any local optimum is global optimum

- Whatever solution we find will be the best solution
- Do not need to worry getting stuck in a local optimum

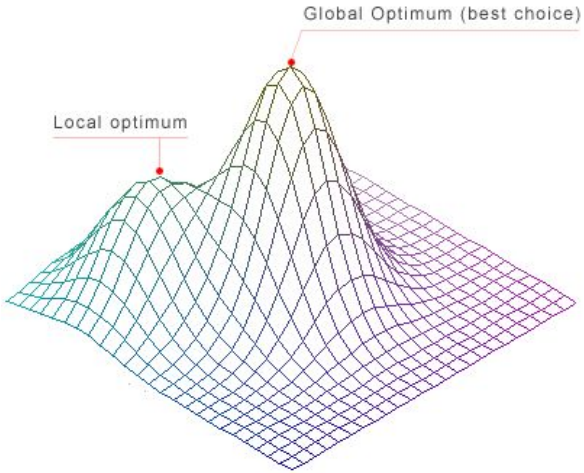


*Image credit: Sarah Nogueira

Optimization for Machine Learning

Given training set: $\{(x_1, y_1), \ldots, (x_n, y_n)\}$

Prediction function: $h(x; \theta)$

Define a loss function: $\mathcal{L}(h(x; \theta), y)$

Find the parameters : θ

which minimizes the empirical risk:

$$\min_{\theta} \frac{1}{n} \sum_{i}^{n} \mathcal{L}(h(x_i; \theta), y_i)$$

Optimization for Machine Learning

We want to minimize the objective function:

$$R(\theta) = \frac{1}{n}\sum_{i}^{n} f_i(\theta) = \frac{1}{n}\sum_{i}^{n} \mathcal{L}(h(x_i;\theta), y_i)$$

- The optimum satisfies:

$$\nabla R(\theta^*) = 0 \qquad \nabla R(\theta) = \left(\frac{\partial R}{\partial \theta_1}, \frac{\partial R}{\partial \theta_2}, \ldots, \frac{\partial R}{\partial \theta_k}\right)$$

- Sometimes the equation has a closed-form solution
  - E.g. linear regression

Gradient Descent Algorithm

We want to minimize the objective function:

$$R(\theta) = \frac{1}{n}\sum_{i}^{n} f_i(\theta) = \frac{1}{n}\sum_{i}^{n} \mathcal{L}(h(x_i;\theta), y_i)$$

(Batch) gradient descent algorithm:

- Initialize the parameters randomly
- For each iteration do:

$$\theta_{k+1} = \theta_k - \eta \nabla R(\theta_k)$$

- Until convergence

Learning rate (a small step)

Gradient Descent Algorithm

Geometric interpretation:

- Gradient is perpendicular to the tangent of the levelset curve
- Given the current point, negative gradient direction decreases the function fastest

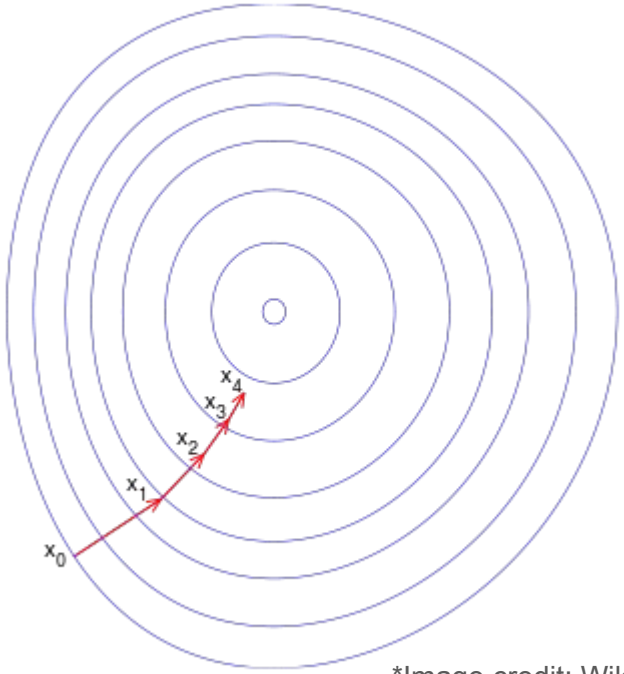


*Image credit: Wikipedia

Alternative interpretation:

- Minimizing the first-order taylor approx of f keep the new point close to the current point

$$f(x^t) + \nabla f(x^t)^T (x - x^t) + \frac{1}{2\eta} \|x - x^t\|_2^2$$

Try to derive grad descent from this

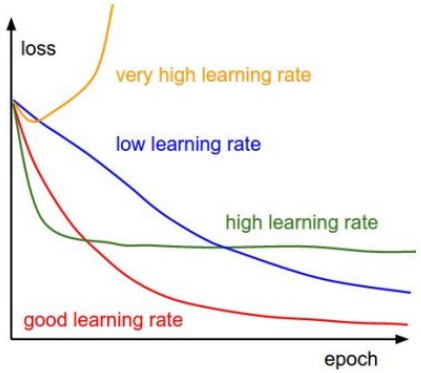
Gradient Descent Algorithm

Learning rate

- Should not be too big (objective will blow up)
- Should not be too small (takes longer to convergent)

Convergence criteria

- Change in objective function is close to zero
- Gradient norm is close to zero
- Validation error starts to increase (early-stopping)



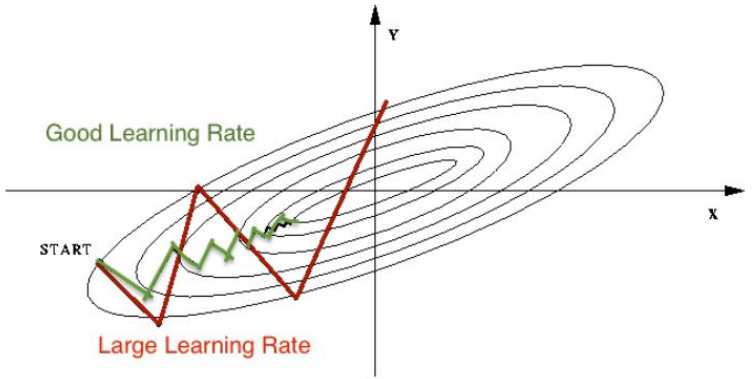


*Image credit: Andrej Karpathy

Practices on Tuning the Learning Rate

Grid search

- Optimize your model with a large learning rate (e.g. 0.1) and then progressively reduce this rate, often by an order of magnitude (0.1, 0.01, 0.001,...)
- The idea is to find the highest stable learning rate

Learning rate decay

- Gradually reduce the learning rate after epochs
- Intuition: supervision signal is strong at early stage and you don’t want to change too much after it’s learnt well

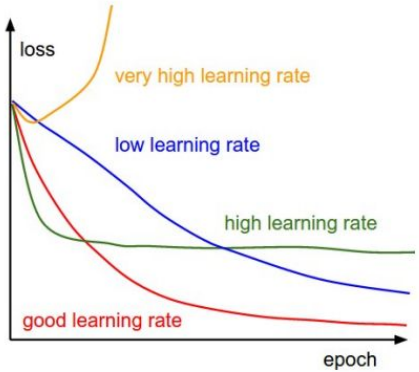


*Image credit: Andrej Karpathy

Gradient Descent Algorithm

(Batch) gradient descent algorithm:

- Initialize the parameters randomly
- For each iteration do:

$$\theta_{k+1} = \theta_k - \eta \nabla R(\theta_k)$$

- Until convergence

Stochastic Gradient Descent Algorithm

- Initialize the parameters randomly
- For each iteration do:
  - Random select a training sample i (or a small subset of training samples)
  - Conduct gradient descent

$$\theta_{k+1} = \theta_k - \eta \nabla f_i(\theta_k)$$

- Until convergence

Intuition: a noisy approximation of the gradient of the whole dataset

Pros: each update requires a small amount of training data, good for training ML algorithms on large-scale dataset.

Gradient Descent with Momentum

- Initialize the parameters randomly
- For each iteration do:
  - Update the momentum:

$$\delta_{k+1} = -\eta \nabla R(\theta_k) + \alpha \delta_k$$

  - Conduct gradient descent

$$\theta_{k+1} = \theta_k + \delta_{k+1}$$

- Until convergence

Pros: accelerate” learning by accumulating some “velocity/momentum” using the past gradients

Nesterov Accelerated Gradient

- Initialize the parameters randomly
- For each iteration do:
  - Update step

$$\delta_{k+1} = -\eta \nabla R(\theta_k + \alpha \delta_k) + \alpha \delta_k$$

  - Conduct gradient descent

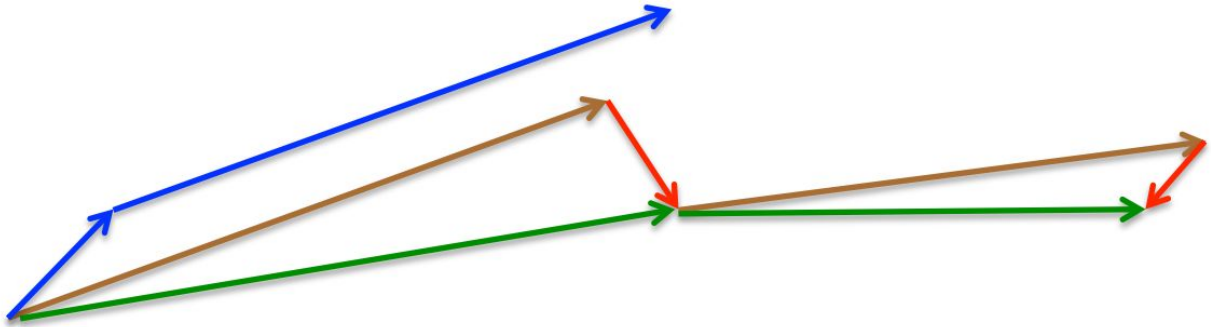
$$\theta_{k+1} = \theta_k + \delta_{k+1}$$

- Until convergence

Intuition: Look into the future to see how much momentum is required.

Nesterov Accelerated Gradient

- First make a big jump in the direction of the previous accumulated gradient.
- Then measure the gradient where you end up and make a correction.



brown vector = jump, red vector = correction, green vector = accumulated gradient

blue vectors = standard momentum

*Slide credit: Geoff Hinton

Learning Rate Schedulers: Adagrad

- Initialize the parameters randomly.
- For each iteration do:
  - Conduct gradient descent for i-th parameter

$$\theta_{k+1,i} = \theta_{k,i} - \frac{\eta}{\sqrt{G_{k,i} + \epsilon}} \cdot \nabla R(\theta_{k,i})$$

$$G_{k,i} = G_{k-1,i} + \left(\nabla R(\theta_{k,i})\right)^2$$

- Until convergence

Intuition: It increases the learning rate for more sparse features and decreases the learning rate for less sparse ones, according to the history of the gradient

Learning Rate Schedulers: RMSprop/Adadelta

- Initialize the parameters randomly. γ is usually set to 0.9
- For each iteration do:
  - Conduct gradient descent for i-th parameter

$$\theta_{k+1,i} = \theta_{k,i} - \frac{\eta}{\sqrt{G_{k,i} + \epsilon}} \cdot \nabla R(\theta_{k,i})$$

$$G_{k,i} = \gamma\, G_{k-1,i} + (1-\gamma)\left(\nabla R(\theta_{k,i})\right)^2$$

- Until convergence

Intuition: Unlike Adagrad, the denominator places a significant weight on the most recent gradient. This also helps avoid decreasing learning rate too much.

Learning Rate Schedulers: Adam

- Initialize the parameters randomly.
- For each iteration do:
  - Conduct gradient descent for i-th parameter

$$\theta_{k+1,i} = \theta_{k,i} - \frac{\eta \cdot \hat{m}_{k,i}}{\hat{G}_{k,i} + \epsilon}$$

$$G_{k,i} = \gamma\, G_{k-1,i} + (1-\gamma)\left(\nabla R(\theta_{k,i})\right)^2 \qquad m_{k,i} = \alpha\, m_{k-1,i} + (1-\alpha)\nabla R(\theta_{k,i})$$

  - $\hat{m}_{k,i}$, $\hat{G}_{k,i}$ are bias corrected forms of $m_{k,i}$, $G_{k,i}$ respectively
- Until convergence

Learning Rate Schedulers: Adam

author

- Initialize the parameters randomly.
- For each iteration do:
  - Conduct gradient descent for i-th parameter

$$\theta_{k+1,i} = \theta_{k,i} - \frac{\eta \cdot \hat{m}_{k,i}}{\hat{G}_{k,i} + \epsilon}$$

$$G_{k,i} = \gamma\, G_{k-1,i} + (1-\gamma)\left(\nabla R(\theta_{k,i})\right)^2 \qquad m_{k,i} = \alpha\, m_{k-1,i} + (1-\alpha)\nabla R(\theta_{k,i})$$

  - $\hat{m}_{k,i}$, $\hat{G}_{k,i}$ are bias corrected forms of $m_{k,i}$, $G_{k,i}$ respectively
- Until convergence

Adam: A method for stochastic optimization
D Kingma, J Ba
International Conference on Learning Representation

1461 2015

Comparison

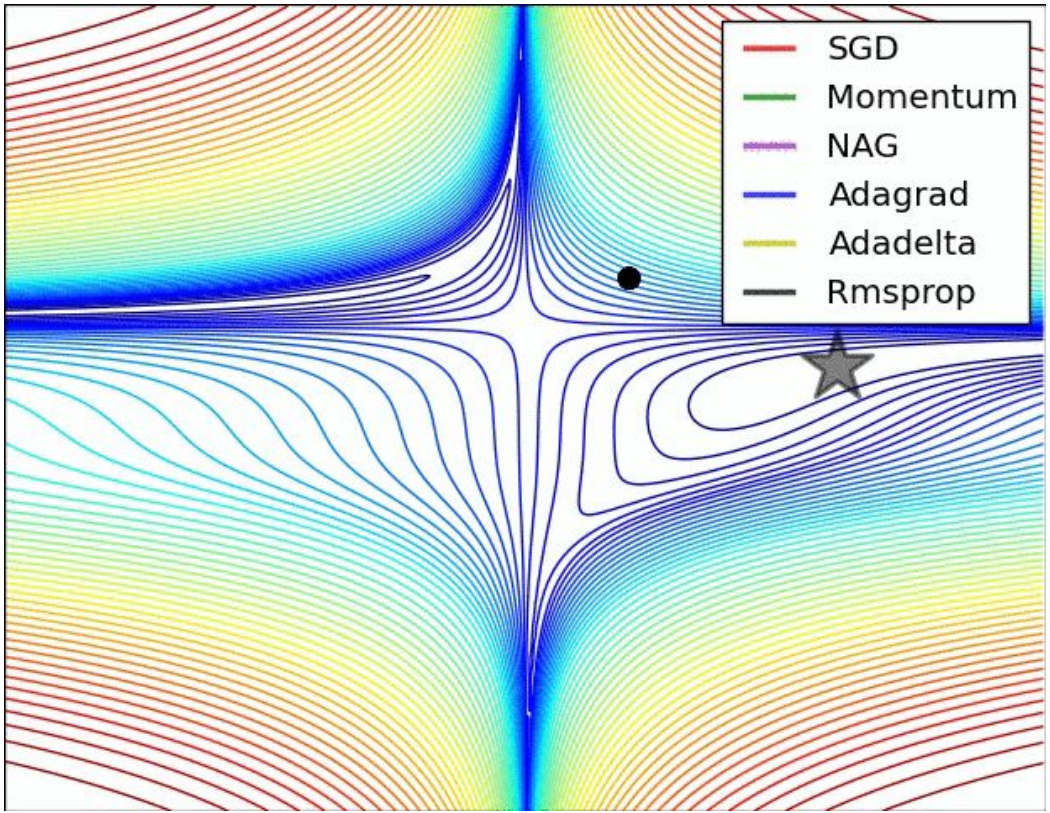


*Image credit: Sebastian Ruder

Checkgrad

- Great tool to debug your implementation
- Finite-approximation to compute gradient according to its definition

$$\frac{\partial f}{\partial \theta_i} = \frac{f(\theta_1, \ldots, \theta_i + \epsilon, \ldots) - f(\theta_1, \ldots, \theta_i - \epsilon, \ldots)}{2\epsilon}$$

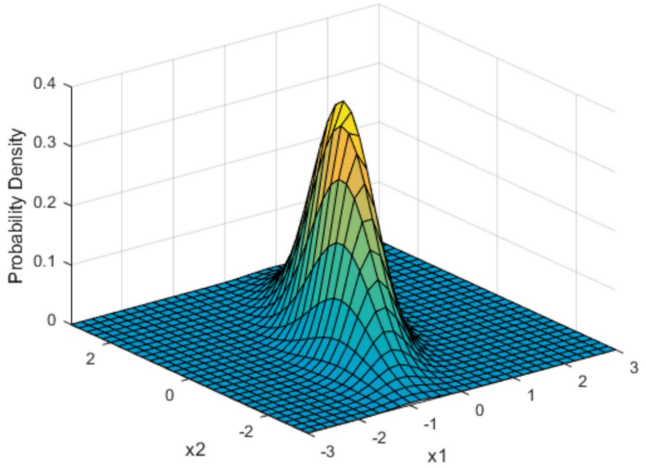
- https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.check_grad.html

Multivariate Gaussian Distribution

- For a D-dimensional vector **x**, the multivariate Gaussian distribution takes the form

$$\mathcal{N}(\mathbf{x}|\mu,\Sigma) = \frac{1}{(2\pi)^{D/2}}\frac{1}{|\Sigma|^{1/2}}\exp\left[-\frac{1}{2}(\mathbf{x}-\mu)^{\top}\Sigma^{-1}(\mathbf{x}-\mu)\right]$$

  - where μ is a D-dimensional mean vector
  - Σ is the DxD dimensional covariance matrix
  - $|\Sigma|$ is the determinant of Σ



Multivariate Gaussian Distribution

- Maximum likelihood estimation of the multivariate Gaussian parameters:

$$l(\mu, \Sigma \mid \mathcal{D}) = -\frac{N}{2} \log |\Sigma| - \frac{1}{2} \sum_{i=1}^{N} (x_i - \mu)^T \Sigma^{-1} (x_i - \mu).$$

$$\frac{\partial l}{\partial \mu} = \sum_{i=1}^{N} (x_i - \mu)^T \Sigma^{-1} \quad \rightarrow \quad \hat{\mu} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

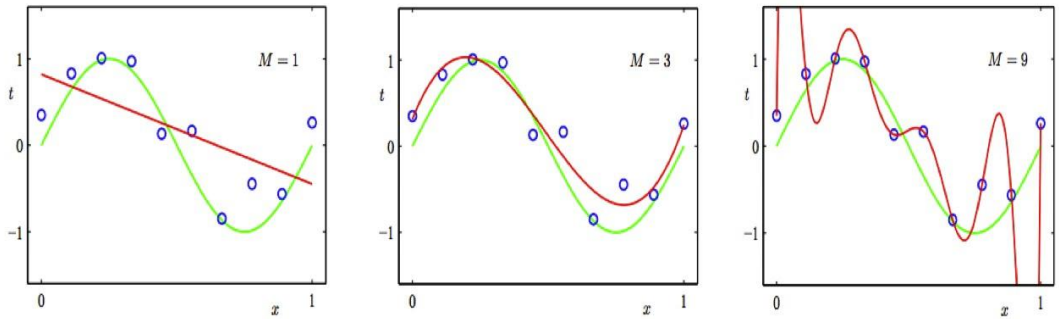
$$\frac{\partial l}{\partial \Sigma^{-1}} = \frac{N}{2} \Sigma - \frac{1}{2} \sum_{n} (x_n - \mu)(x_n - \mu)^T \quad \rightarrow \quad \hat{\Sigma}_{ML} = \frac{1}{N} \sum_{n} (x_n - \hat{\mu}_{ML})(x_n - \hat{\mu}_{ML})^T$$

Eq. 62 https://ece.uwaterloo.ca/~ece603/Matrix-Cookbook.pdf

Underfitting vs Overfitting

- Underfitting: the model is too simple, both training and test errors are large
  - The model does not capture the underlying trend of the data
- Overfitting: models the training data too well
  - The model learns the detail and noise in the training data that would hurt the performance on unseen data

Regression:



predictor too inflexible:
cannot capture pattern

predictor too flexible:
fits noise in the data

*Image credit: Victor Lavrenco

Underfitting vs Overfitting

- Underfitting: the model is too simple, both training and test errors are large
  - The model does not capture the underlying trend of the data
- Overfitting: models the training data too well
  - The model learns the detail and noise in the training data that would hurt the performance on unseen data

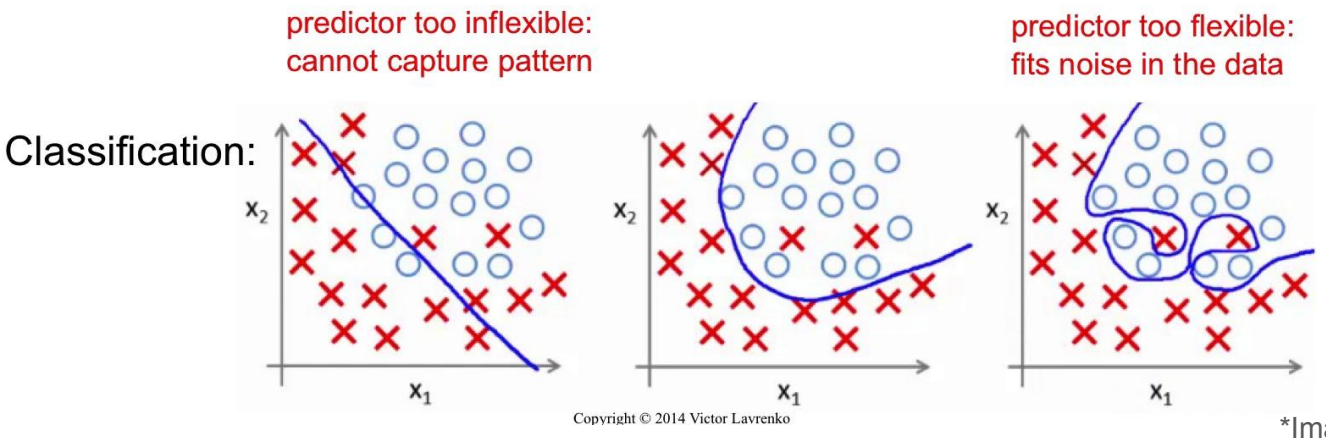
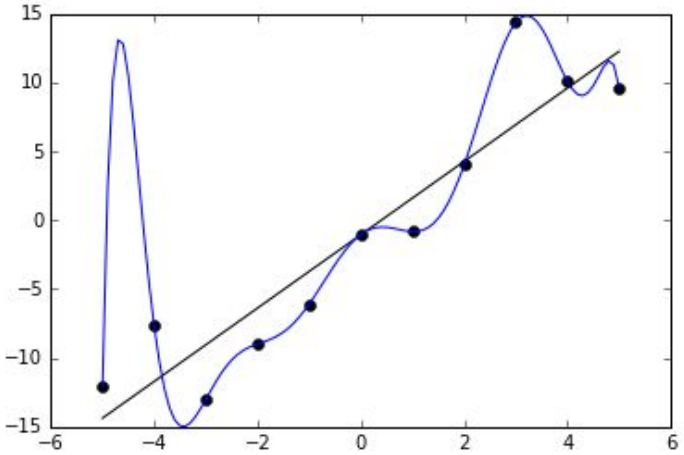
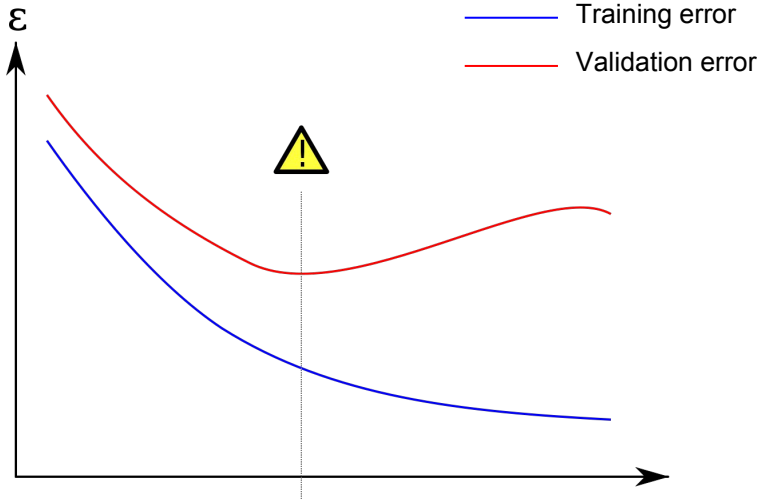


*Image credit: Victor Lavrenco

Underfitting vs Overfitting

- Overfitting: models the training data too well
- Learns the detail and noise in the training data that would hurt the performance on unseen data





Underfitting vs Overfitting

- Generalization error:
  - Performance on unseen data
- Overcome overfitting:
  - Cross-validation
  - Regularization (weight decay, dropout, prior distribution in bayesian, etc.)
  - More training data (data augmentation etc.)
  - Early stopping
  - Less complex models (Occam Razor)