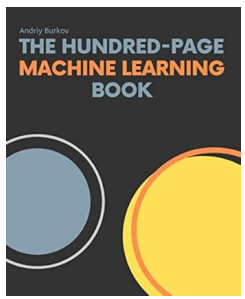
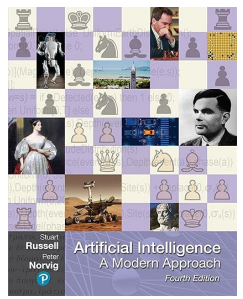
Lesson 02.4: ML Techniques - Model Learning

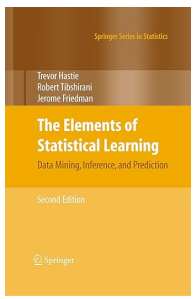
Instructor: Dr. GP Saggese - gsaggese@umd.edu

References:

- Burkov: *"The Hundred-Page Machine Learning Book"* (2019)
- Russell et al.: *"Artificial Intelligence: A Modern Approach"* (4th ed, 2020)
- Hastie et al.: *"The Elements of Statistical Learning"* (2nd ed, 2009)



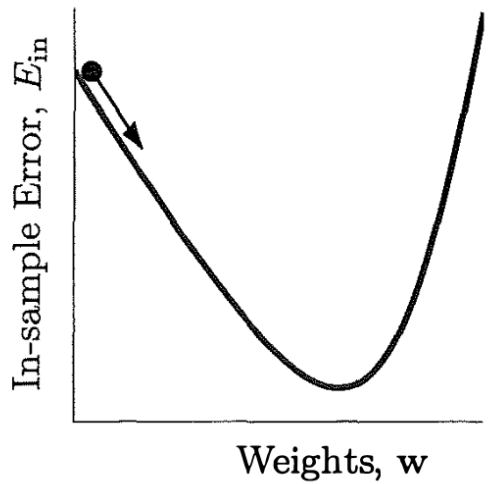




- ***Learning Algorithms***

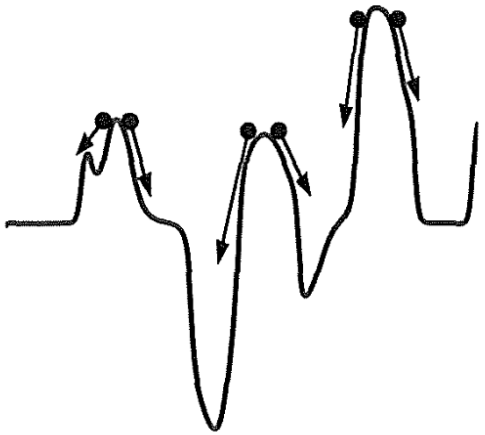
The Problem of Minimizing a Function

- **Goal**: minimize scalar function $J(\underline{\boldsymbol{w}})$ of n-variables $\underline{\boldsymbol{w}}$
  - E.g., in-sample error $E_{in}(\underline{\boldsymbol{w}})$
- **Solutions**:
  1. Analytical solution
     - Impose the gradient of $J(\underline{\boldsymbol{w}})$ to equal 0
     - Find a closed-form solution for $\underline{\boldsymbol{w}}^*$
  2. Numerical solution:
     - Use an iterative method to update $\underline{\boldsymbol{w}}$ to reach the minimum value of $J(\underline{\boldsymbol{w}})$
     - E.g., gradient descent
     - It works even if there is an analytical solution



Gradient Descent: Intuition

- **Problem**:
  - You are on a hilly surface and we want to walk down to the bottom of the hill
- **Solution**:
  - At each point:
    - You look around
    - You move a step in the direction where the surface is steepest
  - You keep doing until we reach the bottom
- **Gradient descent**
  - Is a general technique for minimizing twice-differentiable functions
  - In general, converge to a local minimum
  - If $J(\underline{\boldsymbol{w}})$ is convex, converge to the global minimum
    - E.g., logistic regression and linear models



Gradient descent with fixed learning rate (1/3)

- Consider the contour plot of a function
- **Start** from a point $\underline{\boldsymbol{w}}(0)$ (random, the origin, ...)
- **At each step**
  - Move a fixed amount η in weight space (fixed learning rate):

$$\underline{\boldsymbol{w}}(t+1) = \underline{\boldsymbol{w}}(t) + \eta \hat{\underline{\boldsymbol{v}}}$$

    where $\hat{\underline{\boldsymbol{v}}}$ is a unit vector
  - Pick $\hat{\underline{\boldsymbol{v}}}$ to move to a value of $E_{in}(\underline{\boldsymbol{w}})$ as negative as possible
- The **change for E_{in}** is:

$$\begin{aligned}
\Delta E_{in} &= E_{in}(\underline{\boldsymbol{w}}(t+1)) - E_{in}(\underline{\boldsymbol{w}}(t)) \\
&= E_{in}(\underline{\boldsymbol{w}}(t) + \eta \hat{\underline{\boldsymbol{v}}}) - E_{in}(\underline{\boldsymbol{w}}(t)) && \text{(replacing the expression of } \underline{\boldsymbol{w}}(t+1)) \\
&\approx \eta \nabla E_{in}(\underline{\boldsymbol{w}}(t))^T \hat{\underline{\boldsymbol{v}}} + O(\eta^2) && \text{(using Taylor expansion)}
\end{aligned}$$

- Gradient descent keeps only $O(\eta)$ the term and ignores the rest
- Conjugate gradient considers up to $O(\eta^2)$ and ignores higher infinitesimals

Gradient Descent with Fixed Learning Rate (2/3)

- You have:

$$\Delta E_{in} \approx \eta \nabla E_{in}(\underline{\boldsymbol{w}}(t))^T \hat{\underline{\boldsymbol{v}}}$$

- The **minimal value of the scalar product** ΔE_{in}:
  - Happens when $\hat{\underline{\boldsymbol{v}}} = -\frac{\nabla E_{in}(\underline{\boldsymbol{w}}(t))}{\|\nabla E_{in}(\underline{\boldsymbol{w}}(t))\|}$
  - Is $-\eta \|\nabla E_{in}(\underline{\boldsymbol{w}}(t))\|$
- The **change in weights** $\Delta \underline{\boldsymbol{w}}$ is:

$$\begin{aligned} \Delta \underline{\boldsymbol{w}} &= \underline{\boldsymbol{w}}(t+1) - \underline{\boldsymbol{w}}(t) \\ &= \eta \hat{\underline{\boldsymbol{v}}} \\ &= -\eta \frac{\nabla E_{in}(\underline{\boldsymbol{w}}(t))}{\|\nabla E_{in}(\underline{\boldsymbol{w}}(t))\|} \end{aligned}$$

- Called **"gradient descent"** since it descends along the gradient of the function to optimize

Gradient Descent with Fixed Learning Rate (3/3)

- Each j component of the weight $\underline{\boldsymbol{w}}$ is updated with the partial derivative with respect to that coordinate:

$$\underline{\boldsymbol{w}}(t+1) = \underline{\boldsymbol{w}}(t) - \eta \frac{\nabla E_{in}(\underline{\boldsymbol{w}}(t))}{\|\nabla E_{in}(\underline{\boldsymbol{w}}(t))\|}$$

$$w_j(t+1) = w_j(t) - \eta \frac{1}{\|\nabla E_{in}(\underline{\boldsymbol{w}}(t))\|} \frac{\partial E_{in}(\underline{\boldsymbol{w}})}{\partial w_j}$$

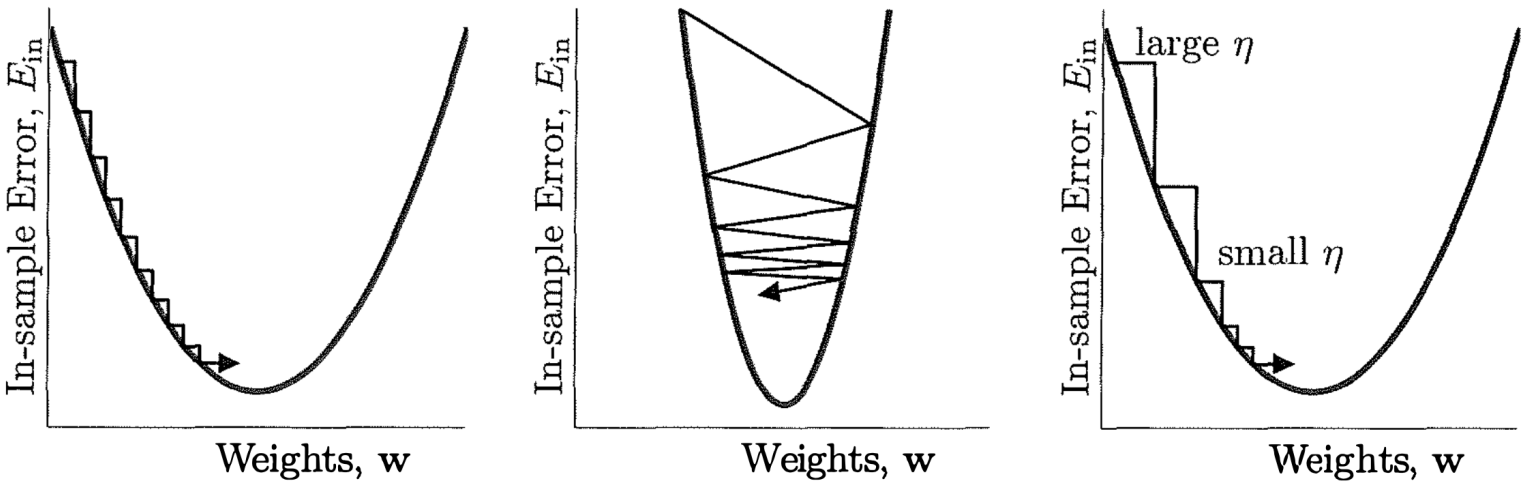
  - The update of all components should be **simultaneous**, i.e., computed at once
- A step of the optimization when we update the solution (weights) is called **epoch**

Gradient Descent: Stopping Criteria

- **In theory**, stop when $\Delta E_{in} = \underline{\mathbf{0}}$
  - Numerically, this might not occur
- **In practice**, stop when:
  - Variation of E_{in} is smaller than threshold $\Delta E_{in} < \theta$
  - Reached a certain number of iterations
- **Monitoring gradient descent**
  - In theory, only compute derivatives of function $J(\underline{\boldsymbol{w}})$ to optimize
  - In practice, monitor progress by recomputing cost function $J(\underline{\boldsymbol{w}})$ periodically to ensure it decreases

Gradient Descent with Fixed Learning Rate

- Consider a 1D convex function
  - If η **is too small**:
    - Linear approximation of E_{in} is effective
    - Many steps needed to converge to minimum
  - If η **is too large**:
    - Linear approximation fails (higher terms affect values)
    - It "bounces around"



η too small | η too large | variable η | just right

- **Idea**: vary learning rate η during gradient descent
  - Smaller learning rates may find a better minimum
  - Reduce η as a function of iterations
  - **Cons**: introduces an additional parameter to tune

Gradient Descent with Variable Learning Rate

- In **gradient descent with fixed learning rate** (constant change in weight space), use:

$$\Delta \underline{\boldsymbol{w}} = -\eta \frac{\nabla J}{\|\nabla J\|}$$

- **To converge quickly:**
  - Move fast (large change) in weight space (large η) when surface is steep (large gradient)
  - Move slow (small change) in weight space (small η) near minimum to avoid bouncing (small gradient)
  - Ideally, η should increase with slope: $\eta \propto \|\nabla J\|$
- **Gradient descent with variable learning rate**:

$$\Delta \underline{\boldsymbol{w}} = -\eta \nabla J$$

Feature Scaling in Gradient Descent

- **Problem**: Different gradient components have different errors due to numerical approximation, causing bouncing
  - Unscaled features lead to slow, unstable convergence
  - E.g., feature ranging from 1 to 1000 vs. 0.01 to 1 causes inefficient updates
- **Solution**: Converge faster by scaling features to the same range
  - Min-max scaling
  - Standardization (transforms feature values to mean 0, standard deviation 1)

Issues with Batch Gradient Descent

- Consider squared error with n samples:

$$E_{in}(\underline{\boldsymbol{w}}) \triangleq \frac{1}{n}\sum_{i=1}^{n}(h_{\underline{\boldsymbol{w}}}(\underline{\boldsymbol{x}}_i) - y_i)^2$$

- **Batch Gradient Descent (BGD)** updates each weight component:

$$\underline{\boldsymbol{w}}(t+1) = \underline{\boldsymbol{w}}(t) - \eta\frac{\nabla E_{in}}{\|\nabla E_{in}\|}$$

  - Coordinate update for squared error:

$$w_j(t+1) = w_j(t) - \eta\frac{2}{n}\sum_{i=1}^{n}(h_{\underline{\boldsymbol{w}}}(\underline{\boldsymbol{x}}_i) - y_i)\frac{\partial h_{\underline{\boldsymbol{w}}}(\underline{\boldsymbol{x}}_i)}{\partial w_j}$$

  - **Cons**: gradient descent with many training examples (e.g., $N = 10^6$)
    - Is computationally expensive, requiring gradient evaluation from all examples for one update
    - Requires storing all data in memory

Stochastic Gradient Descent

- Idea of **Stochastic Gradient Descent (SGD)**: update the weights only for one training example picked at random
- **Algorithm**
  - Pick one $(\underline{\boldsymbol{x}}_n, y_n)$ at a time from the available examples
  - Compute $\nabla e(h(\underline{\boldsymbol{x}}_n), y_n)$ to update the weights: $\Delta \underline{\boldsymbol{w}} = -\eta \nabla e$
  - Update the weight considering only one random example:

$$w_j(t+1) = w_j(t) - \eta \frac{2}{n}(h_{\underline{\boldsymbol{w}}}(\underline{\boldsymbol{x}}_t) - y_t)\frac{\partial h_{\underline{\boldsymbol{w}}}(\underline{\boldsymbol{x}}_t)}{\partial w_j}$$

- **Cons**: in Stochastic Gradient Descent
  - Path in weight space is random
  - Oscillates around local minimum
- **Why SGD works**
  - ∇e is a function of a random var $\underline{\boldsymbol{x}}_n$
  - The average direction of SGD is the same direction as batch version

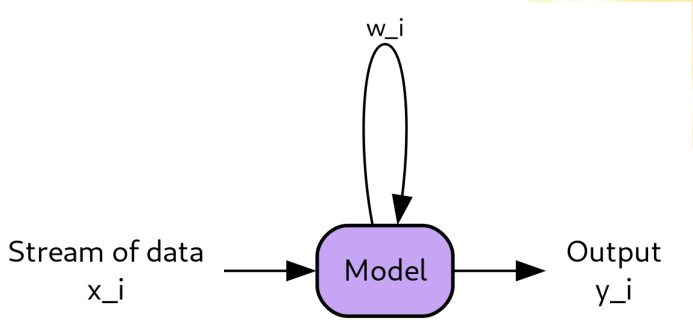
$$\mathbb{E}[\nabla e] = \frac{1}{N}\sum \nabla e(h(\underline{\boldsymbol{x}}_n), y_n) = \nabla \frac{1}{N}\sum e(h(\underline{\boldsymbol{x}}_n), y_n) = \nabla E_{in}$$

Mini-Batch Gradient Descent

- **Mini-Batch Gradient Descent** brings together characteristics of both Batch and Stochastic Gradient Descent
  - Use b examples to make an update to the current weight
  - b represents the batch size
  - A common choice is $b = 32$ or $b = 64$
- Mini-batch GD offers a balance between SGD noisiness and full-batch approaches, using small, random data samples for updates

On-Line Learning and Gradient Descent

- In many applications, **continuous stream** of training examples $\rightarrow$ requires updating the model
  - In real-time systems, adapt model with new data points without full retraining
  - Handle variation in underlying process dynamics
- Stochastic GD and mini-batch GD **suitable for online learning**
  - Update model one example at a time
  - Discard examples for "compressed" model representation
    - Useful for large data streams where storing every data point is impractical
    - E.g., in stock market prediction
    - E.g., in training a language model on live chat data, discard older conversations after updates



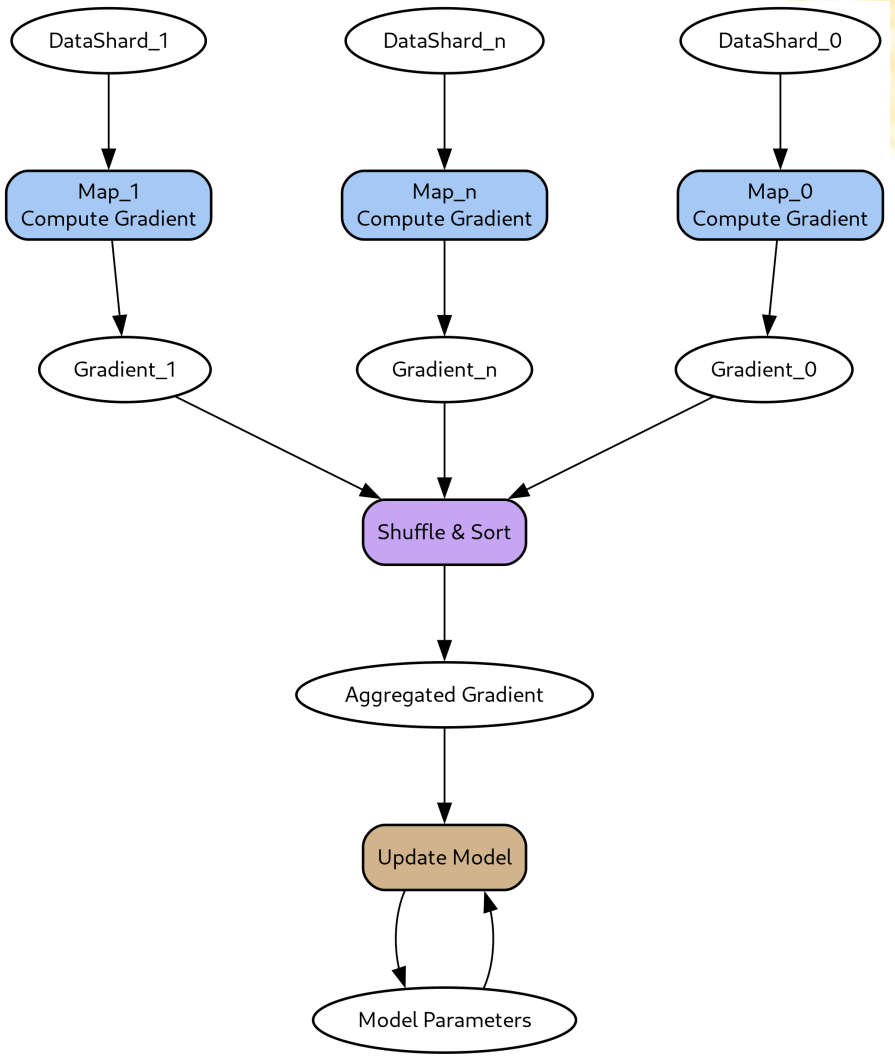
SGD vs BGD vs Mini-Batch

- To update the weights:
  - **BGD (batch gradient descent)** uses all the training examples
  - **SGD (stochastic gradient descent)** uses a single random training example
  - **Mini-batch GD** uses a random subset of training examples

Aspect	**Batch Gradient Descent**	**Stochastic Gradient Descent**
Computation	Uses all examples	One example at a time
Memory	Requires all examples in memory	Require less memory
Randomization	More likely to terminate in flat regions	Avoid local minima due to randomness
Regularization	No implicit regularization	Oscillations act as regularization
Parallelization	Can be parallelized	Less parallel-friendly
Online Learning	Not suitable	Suitable for online learning

Map-Reduce for Batch Gradient Descent

- Batch GD (and many learning algorithms) can be expressed in map-reduce
- **In map-reduce**
  - **Map step**: use k machines to parallelize summation
  - **Reduce step**: send k partial sums to a single node to accumulate result



Coordinate Descend

- **Idea**: minimize $J(x_0, ..., x_n)$ by optimizing along one direction x_i at a time
  - Instead of computing all derivatives
- **Algorithm**
  - Pick a random starting point $\underline{\boldsymbol{w}}(0)$
  - Pick a random order for the coordinates $\{x_i\}$
  - Find the minimum along the current coordinate (1D optimization problem)
  - Move to the next coordinate x_{i+1}
  - The sequence of $\underline{\boldsymbol{w}}(t)$ is decreasing
  - A minimum is found if there is no improvement after one cycle of scanning all coordinates
  - The minimum is local

Gradient Descent vs Pseudo-Inverse

- For linear models, you can use pseudo-inverse or gradient descent to find optimal $\underline{\boldsymbol{w}}^*$
- **Gradient descent**
  - Choose learning rate η
  - Requires many iterations
  - Monitor stopping criteria, oscillations
  - Effective for many features P
- **Pseudo-inverse**
  - No parameter selection
  - Converges in one iteration
  - Computes a $P \times P$ matrix $(\underline{\underline{\boldsymbol{X}}}^T \underline{\underline{\boldsymbol{X}}})^{-1}$
    - Complexity of matrix inversion $O(P^3)$
    - For $P \approx 10,000$, prefer gradient descent