Machine Learning is Optimization

• Parametric ML involves minimizing an **objective function** $J(w)$:
  – Also called **cost function, loss function, or error function**.
  – Want to find $\hat{w} = \arg\min_w J(w)$

• Numerical optimization procedure:
  1. Start with some guess for $w^0$, set $\tau = 0$.
  2. **Update** $w^\tau$ to $w^{\tau+1}$ such that $J(w^{\tau+1}) \leq J(w^\tau)$.
  3. Increment $\tau = \tau + 1$.
  4. Repeat from 2 until $J$ cannot be improved anymore.
Gradient-based Optimization

• How to update \( w^\tau \) to \( w^{\tau+1} \) such that \( J(w^{\tau+1}) \leq J(w^\tau) \)?

• Move \( w \) in the direction of **steepest descent**:

\[
w^{\tau+1} = w^\tau + \eta g
\]

- \( g \) is the direction of steepest descent, i.e. direction along which \( J \) decreases the most.
- \( \eta \) is the learning rate and controls the magnitude of the change.
Gradient-based Optimization

- Move \( \mathbf{w} \) in the direction of **steepest descent**:
  \[
  \mathbf{w}^{\tau+1} = \mathbf{w}^\tau + \eta \mathbf{g}
  \]

- What is the direction of steepest descent of \( J(\mathbf{w}) \) at \( \mathbf{w}^\tau \)?
  - The gradient \( \nabla J(\mathbf{w}) \) is in the direction of steepest ascent.
  - Set \( \mathbf{g} = -\nabla J(\mathbf{w}) \Rightarrow \) the **gradient descent** update:
    \[
    \mathbf{w}^{\tau+1} = \mathbf{w}^\tau - \eta \nabla J(\mathbf{w}^\tau)
    \]
Gradient Descent Algorithm

- Want to minimize a function \( J : \mathbb{R}^n \rightarrow \mathbb{R} \).
  - \( J \) is differentiable and convex.
  - Compute gradient of \( J \) i.e. *direction of steepest increase*:
    \[
    \nabla J(w) = \left[ \frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \ldots, \frac{\partial J}{\partial w_n} \right]
    \]

1. Set learning rate \( \eta = 0.001 \) (or other small value).
2. Start with some guess for \( w^0 \), set \( \tau = 0 \).
3. Repeat for epochs \( E \) or until \( J \) does not improve:
4. \( \tau = \tau + 1 \).
5. \( w^{\tau+1} = w^\tau - \eta \nabla J(w^\tau) \)
Gradient Descent: Large Updates
Gradient Descent: Small Updates

Cost

Learning step

Minimum

Random initial value

θ

https://www.safaribooksonline.com/library/view/hands-on-machine-learning
The Learning Rate

1. Set **learning rate** $\eta = 0.001$ (or other small value).
2. Start with some guess for $w^0$, set $\tau = 0$.
3. Repeat for epochs E or until J does not improve:
   4. $\tau = \tau + 1$.
   5. $w^{\tau+1} = w^\tau - \eta \nabla J(w^\tau)$

**How big should the **learning rate** be?**
  - If learning rate too small => slow convergence.
  - If learning rate too big => oscillating behavior => may not even converge.
Learning Rate too Small
Learning Rate too Large
The Learning Rate

• How big should the learning rate be?
  – If learning rate too big => oscillating behavior.
  – If learning rate too small => hinders convergence.

○ Use line search (backtracking line search, conjugate gradient, …).
○ Use second order methods (Newton’s method, L-BFGS, …).
  • Requires computing or estimating the Hessian.

○ Use a simple learning rate annealing schedule:
  – Start with a relatively large value for the learning rate.
  – Decrease the learning rate as a function of the number of epochs or as a function of the improvement in the objective.

○ Use adaptive learning rates:
  • Adagrad, Adadelta, RMSProp, Adam.
Gradient Descent: Nonconvex Objective

- Cost
  - Local minimum
  - Global minimum
  - Saddle point
  - Plateau
Convex Multivariate Objective
Gradient Step and Contour Lines
Gradient Descent: Nonconvex Objectives
Gradient Descent & Plateaus
Gradient Descent & Saddle Points
Gradient Descent & Ravines
Gradient Descent & Ravines

- **Ravines** are areas where the surface curves much more steeply in one dimension than another.
  - Common around local optima.
  - GD oscillates across the slopes of the ravines, making slow progress towards the local optimum along the bottom.

- Use **momentum** to help accelerate GD in the relevant directions and dampen oscillations:
  - Add a fraction of the past **update vector** to the current update vector.
    - The momentum term increases for dimensions whose previous gradients point in the same direction.
    - It reduces updates for dimensions whose gradients change sign.
    - Also reduces the risk of getting stuck in local minima.
Gradient Descent & Momentum

Vanilla Gradient Descent:

\[ v^{\tau+1} = \eta \nabla J(w^{\tau}) \]
\[ w^{\tau+1} = w^{\tau} - v^{\tau+1} \]

Gradient Descent w/ Momentum:

\[ v^{\tau+1} = \gamma v^{\tau} + \eta \nabla J(w^{\tau}) \]
\[ w^{\tau+1} = w^{\tau} - v^{\tau+1} \]

\( \gamma \) is usually set to 0.9 or similar.
Momentum & Nesterov Accelerated Gradient

GD with Momentum:
\[
\begin{align*}
v^{\tau+1} &= \gamma v^{\tau} + \eta \nabla J(w^{\tau}) \\
w^{\tau+1} &= w^{\tau} - v^{\tau+1}
\end{align*}
\]

Nesterov Accelerated Gradient:
\[
\begin{align*}
v^{\tau+1} &= \gamma v^{\tau} + \eta \nabla J(w^{\tau} - \gamma v^{\tau}) \\
w^{\tau+1} &= w^{\tau} - v^{\tau+1}
\end{align*}
\]

By making an anticipatory update, NAGs prevents GD from going too fast => significant improvements when training RNNs.
Gradient Descent Optimization Algorithms

- **Momentum.**
- **Nesterov Accelerated Gradient (NAG).**
- Adaptive learning rates methods:
  - Idea is to perform larger updates for infrequent params and smaller updates for frequent params, by accumulating previous gradient values for each parameter.
    - **Adagrad.**
    - **Adadelta.**
    - **RMSProp.**
    - **Adaptive Moment Estimation (Adam)**
• Adagrad, RMSprop, Adadelta, and Adam are very similar algorithms that do well in similar circumstances.
  – Insofar, **Adam** might be the best overall choice.
Variants of Gradient Descent

\[ w^{\tau + 1} = w^\tau - \eta \nabla J(w^\tau) \]

- Depending on how much data is used to compute the gradient at each step:
  - **Batch gradient descent**: Use all the training examples.
  - **Stochastic gradient descent** (SGD): Use one training example, update after each.
  - **Minibatch gradient descent**: Use a constant number of training examples (minibatch).
Batch Gradient Descent

- Sum-of-squares error:

\[ J(w) = \frac{1}{2N} \sum_{n=1}^{N} (h_w(x^{(n)}) - t_n)^2 \]

\[ w^{\tau+1} = w^{\tau} - \eta \nabla J(w^{\tau}) \]
Stochastic Gradient Descent

- Sum-of-squares error:

\[
J(w) = \frac{1}{2N} \sum_{n=1}^{N} (h_w(x^{(n)}) - t_n)^2 = \frac{1}{2N} \sum_{n=1}^{N} J(w^\tau, x^{(n)})
\]

\[
w^{\tau+1} = w^\tau - \eta \nabla J(w^\tau, x^{(n)})
\]

\[
w^{\tau+1} = w^\tau - \eta (h_w(x^{(n)}) - t_n) x^{(n)}
\]

- Update parameters \( w \) after each example, sequentially:
  
  \( \Rightarrow \) the least-mean-square (LMS) algorithm.
Batch GD vs. Stochastic GD

- Accuracy:
- Time complexity:
- Memory complexity:
- Online learning:
Batch GD vs. Stochastic GD
Pre-processing Features

- Features may have very different scales, e.g. $x_1 = \text{rooms}$ vs. $x_2 = \text{size in sq ft.}$
  - **Right (different scales):** GD goes first towards the bottom of the bowl, then slowly along an almost flat valley.
  - **Left (scaled features):** GD goes straight towards the minimum.

![Diagram showing different optimization paths for scaled and unscaled features](image)
Feature Scaling

• Scaling between $[0, 1]$ or $[-1, +1]$:
  – For each feature $x_j$, compute $\min_j$ and $\max_j$ over the training examples.
  – Scale $x_{(n)}^j$ as follows:

• Scaling to standard normal distribution:
  – For each feature $x_j$, compute sample $\mu_j$ and sample $\sigma_j$ over the training examples.
  – Scale $x_{(n)}^j$ as follows:
Implementation: Gradient Checking

• Want to minimize $J(\theta)$, where $\theta$ is a scalar.

• Mathematical definition of derivative:

$$\frac{d}{d\theta}J(\theta) = \lim_{\varepsilon \to \infty} \frac{J(\theta + \varepsilon) - J(\theta - \varepsilon)}{2\varepsilon}$$

• Numerical approximation of derivative:

$$\frac{d}{d\theta}J(\theta) \approx \frac{J(\theta + \varepsilon) - J(\theta - \varepsilon)}{2\varepsilon} \quad \text{where } \varepsilon = 0.0001$$
Implementation: Gradient Checking

- If $\theta$ is a vector of parameters $\theta_i$,
  - Compute numerical derivative with respect to each $\theta_i$.
  - Aggregate all derivatives into numerical gradient $G_{\text{num}}(\theta)$.

- Compare numerical gradient $G_{\text{num}}(\theta)$ with implementation of gradient $G_{\text{imp}}(\theta)$:

  $$\frac{\|G_{\text{num}}(\theta) - G_{\text{imp}}(\theta)\|}{\|G_{\text{num}}(\theta) + G_{\text{imp}}(\theta)\|} \leq 10^{-6}$$
Gradient Descent vs. Normal Equations

- **Gradient Descent:**
  - Need to select learning rate $\eta$.
  - May need many iterations:
    - Can do *Early Stopping* on validation data for regularization.
    - Scalable when number of training examples $N$ is large.

- **Normal Equations:**
  - No iterations $\Rightarrow$ easy to code.
  - Computing $(X^TX)^{-1}$ has cubic time complexity $\Rightarrow$ slow for large $N$.
  - $X^TX$ may be singular:
    1. Redundant (linearly dependent) features.
    2. #features $>$ #examples $\Rightarrow$ do *feature selection* or *regularization*. 