Neural Network Optimization
Machine Learning Problems

- Given data:

\[ \{(x_i, y_i)\}_{i=1}^{n} \quad x_i \in \mathbb{R}^d \quad y_i \in \mathbb{R} \]

- Learning a model’s parameters:

\[
\min_{\mathbf{w}} \sum_{i=1}^{n} \ell_i(\mathbf{w})
\]

  Logistic Loss: \( \ell_i(\mathbf{w}) = \log(1 + \exp(-y_i x_i^T \mathbf{w})) \)

  Squared error Loss: \( \ell_i(\mathbf{w}) = (y_i - x_i^T \mathbf{w})^2 \)
Machine Learning Problems

- **Given data:**

  \[ \{(x_i, y_i)\}_{i=1}^{n} \quad x_i \in \mathbb{R}^d \quad y_i \in \mathbb{R} \]

- **Learning a model’s parameters:**

  \[ \sum_{i=1}^{n} l_i(w) \]

  Logistic Loss: \[ l_i(w) = \log(1 + \exp(-y_i x_i^T w)) \]

  Squared error Loss: \[ l_i(w) = (y_i - x_i^T w)^2 \]

  Gradient Descent:

  \[ w_{t+1} = w_t - \eta \nabla_w \left( \frac{1}{n} \sum_{i=1}^{n} l_i(w) \right) \bigg|_{w=w_t} \]

  0 initialization for \( w_0 \)

  or random init \( w_0 \sim \mathcal{D}_w \)
Gradient Descent

Initialize: $w_0 = 0$

for $t = 1, 2, \ldots$

$$w_{t+1} = w_t - \eta \nabla f(w_t)$$
Sub-Gradient Descent

Initialize: \( w_0 = 0 \)
for \( t = 1, 2, \ldots \)

Find any \( g_t \) such that

\[
f(y) \geq f(w_t) + g_t^T(y - w_t)
\]

\[
w_{t+1} = w_t - \eta g_t
\]

\( g \) is a subgradient at \( x \) if

\[
f(y) \geq f(x) + g^T(y - x)
\]

Convex Function

Non-convex Function
Machine Learning Problems

- **Given data:**
  \[
  \{(x_i, y_i)\}_{i=1}^n \quad x_i \in \mathbb{R}^d \quad y_i \in \mathbb{R}
  \]

- **Learning a model’s parameters:**
  \[
  \sum_{i=1}^n l_i(w) \text{ time complexity } O(n)
  \]

**Gradient Descent:**
\[
  w_{t+1} = w_t - \eta \nabla_w \left( \frac{1}{n} \sum_{i=1}^n l_i(w) \right) \bigg|_{w=w_t}
\]

**Stochastic Gradient Descent:**
\[
  w_{t+1} = w_t - \eta \nabla_w l_{I_t}(w) \bigg|_{w=w_t}
\]

**SGD:**
\[
  \frac{1}{\delta^2}
\]

**Strongly convex:**
\[
  C(\delta) = \frac{1}{2} \log \left( \frac{1}{\delta} \right)
\]

**Expected:**
\[
  \mathbb{E} \left[ l_{I_t}(w) \right] = \nabla \mathcal{L} \left( \frac{1}{n} \sum_{i=1}^n l_i(w) \right)
\]

**I_t** drawn uniform at random from \(\{1, \ldots, n\}\)
Mini-batch SGD

Instead of one iterate, average $B$ stochastic gradient together

Advantages:
- de-noises gradient
- Matrix computations
- Parallelization

$$\frac{1}{B} \sum_{i=1}^{B} \nabla \mathcal{L}(\theta)$$
Gradient Computation on a Graph

Naive computation: node by node

\[ \frac{\partial L}{\partial \Theta^{(1)}} = O(L), \quad \sigma(l^2) \]
A brief history

- **Back propagation**: the workhorse for training neural networks. An algorithm that for a network with V nodes and E edges calculates that gradient in **linear time** $O(V+E)$.

- The name was introduced by Rumelhart, Hinton, Williams ’86. Same idea was rediscovered multiple times. Also mentioned in Werbos’ thesis ‘74 in the context of neural networks.

- **Control theory**: Kelly ’60, Bryson ’61 ([dynamic programming](#))

- **Theoretical computer science**: Baur-Strassen lemma ’83 ([algebraic circuits](#))
\( a^{(1)} = x \)

\( z^{(2)} = \Theta^{(1)} a^{(1)} \)

\( a^{(2)} = g \left( z^{(2)} \right) \)

\( \vdots \)

\( z^{(l+1)} = \Theta^{(l)} a^{(l)} \)

\( a^{(l+1)} = g \left( z^{(l+1)} \right) \)

\( \vdots \)

\( \hat{y} = g \left( \Theta^{(L)} a^{(L)} \right) \)

\[
L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})
\]

\[
g(z) = \frac{1}{1 + e^{-z}}
\]

**Gradient Descent:**

\[
\Theta^{(l)} \leftarrow \Theta^{(l)} - \eta \nabla_{\Theta^{(l)}} L(y, \hat{y}) \quad \forall l
\]
Forward Propagation

\[ a^{(1)} = x \]
\[ z^{(2)} = \Theta^{(1)} a^{(1)} \]
\[ a^{(2)} = g(z^{(2)}) \]
\[ \vdots \]
\[ a^{(l)} = g(z^{(l)}) \]
\[ z^{(l+1)} = \Theta^{(l)} a^{(l)} \]
\[ a^{(l+1)} = g(z^{(l+1)}) \]
\[ \vdots \]
\[ \hat{y} = a^{(L+1)} \]

\[ L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) \]

\[ g(z) = \frac{1}{1 + e^{-z}} \]

\( L \): # of layers
Ignore bias
\( g \): activation func
\( \hat{z} \): pre-activation
Backprop

\[ a^{(1)} = x \]
\[ z^{(2)} = \Theta^{(1)} a^{(1)} \]
\[ a^{(2)} = g \left( z^{(2)} \right) \]
\[ \vdots \]
\[ a^{(l)} = g \left( z^{(l)} \right) \]
\[ z^{(l+1)} = \Theta^{(l)} a^{(l)} \]
\[ a^{(l+1)} = g \left( z^{(l+1)} \right) \]
\[ \vdots \]
\[ \hat{y} = a^{(L+1)} \]

Train by Stochastic Gradient Descent:

Scalar

\[ \Theta^{(l)}_{i,j} \leftarrow \Theta^{(l)}_{i,j} - \eta \frac{\partial L(y, \hat{y})}{\partial \Theta^{(l)}_{i,j}} \]

wait for all \( i, k, l \)

\[ L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) \]
\[ g(z) = \frac{1}{1 + e^{-z}} \]
\[ \delta^{(l+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \]

\( \chi \in \mathbb{R}^d, \Theta^{(1)}, \ldots, \Theta^{(L)} \) parameters to train
\( \Theta^{(l)} \in \mathbb{R}^{m \times d}, \Theta^{(l+1)} \in \mathbb{R}^{m \times m}, \Theta^{(L)} \in \mathbb{R}^{m} \)

\( m \): width
Backprop

\[ a^{(1)} = x \]

\[ z^{(2)} = \Theta^{(1)} \cdot a^{(1)} \]

\[ a^{(2)} = g(z^{(2)}) \]

\[ \vdots \]

\[ a^{(l)} = g(z^{(l)}) \]

\[ z^{(l+1)} = \Theta^{(l)} \cdot a^{(l)} \]

\[ a^{(l+1)} = g(z^{(l+1)}) \]

\[ \vdots \]

\[ \hat{y} = a^{(L+1)} \]

**Train by Stochastic Gradient Descent:**

\[ \Theta^{(l)}_{i,j} \leftarrow \Theta^{(l)}_{i,j} - \eta \frac{\partial L(y, \hat{y})}{\partial \Theta^{(l)}_{i,j}} \]

**Chain Rule:**

\[ z^{(l+1)}_i = \frac{1}{m} \sum_{j=1}^{m} \Theta^{(l)}_{i,j} \cdot a^{(l)}_j \]

\[ \frac{\partial L(y, \hat{y})}{\partial \Theta^{(l)}_{i,j}} = \frac{\partial L(y, \hat{y})}{\partial z^{(l+1)}_i} \cdot \frac{\partial z^{(l+1)}_i}{\partial z^{(l)}_i} \]

\[ \delta^{(l+1)}_i = \frac{\partial L(y, \hat{y})}{\partial z^{(l+1)}_i} \cdot a^{(l)}_j \]

**Logistic Loss Function:**

\[ L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) \]

\[ g(z) = \frac{1}{1 + e^{-z}} \]

\[ \delta^{(l+1)}_i = \frac{\partial L(y, \hat{y})}{\partial z^{(l+1)}_i} \]

\[ a^{(L+1)} = \hat{y} \]
Backprop

\[ a^{(1)} = x \]
\[ z^{(2)} = \Theta^{(1)} a^{(1)} \]
\[ a^{(2)} = g \left( z^{(2)} \right) \]
\[ \vdots \]
\[ a^{(l)} = g \left( z^{(l)} \right) \]
\[ z^{(l+1)} = \Theta^{(l)} a^{(l)} \]
\[ a^{(l+1)} = g \left( z^{(l+1)} \right) \]
\[ \vdots \]
\[ \hat{y} = a^{(L+1)} \]

\[ L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) \]

\[ g(z) = \frac{1}{1 + e^{-z}} \]
\[ \delta^{(l+1)}_i = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \]

\[ \frac{\partial L(y, \hat{y})}{\partial \Theta^{(l)}_{i,j}} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta^{(l)}_{i,j}} =: \delta^{(l+1)}_i \cdot a^{(l)}_j \]

\[ \delta^{(l)}_i = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l)}} = \sum_k \frac{\partial L(y, \hat{y})}{\partial z_k^{(l+1)}} \cdot \frac{\partial z_k^{(l+1)}}{\partial z_i^{(l)}} \cdot \delta^{(l+1)}_k \]

\[ z_k^{(l+1)} = \sum_{u=1}^m \Theta^{(l)}_{ku} \cdot g(z_{u}^{(l)}) \]

\[ \Theta^{(l)}_{ki} \cdot y_{i}^{(l)} \]
Backprop

\[ a^{(1)} = x \]
\[ z^{(2)} = \Theta^{(1)} a^{(1)} \]
\[ a^{(2)} = g(z^{(2)}) \]
\[ \vdots \]
\[ a^{(l)} = g(z^{(l)}) \]
\[ z^{(l+1)} = \Theta^{(l)} a^{(l)} \]
\[ a^{(l+1)} = g(z^{(l+1)}) \]
\[ \vdots \]
\[ \hat{y} = a^{(L+1)} \]

\[ \frac{\partial L(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)} \]

\[ \delta_i^{(l)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l)}} = \sum_k \frac{\partial L(y, \hat{y})}{\partial z_k^{(l+1)}} \cdot \frac{\partial z_k^{(l+1)}}{\partial z_i^{(l)}} \]
\[ = \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)} g'(z_i^{(l)}) \]
\[ = a_i^{(l)} (1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)} \]

\[ L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) \]
\[ g(z) = \frac{1}{1 + e^{-z}} \]
\[ \delta_i^{(l+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \]
Backprop

\[
\begin{align*}
  a^{(1)} & = x \\
  z^{(2)} & = \Theta^{(1)} a^{(1)} \\
  a^{(2)} & = g(z^{(2)}) \\
  & \vdots \\
  a^{(l)} & = g(z^{(l)}) \\
  z^{(l+1)} & = \Theta^{(l)} a^{(l)} \\
  a^{(l+1)} & = g(z^{(l+1)}) \\
  & \vdots \\
  \hat{y} & = a^{(L+1)}
\end{align*}
\]

Recursion / Dynamic Programming

\[
\frac{\partial L(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}
\]

\[
\delta_i^{(l)} = a_i^{(l)} (1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)}
\]

\[
L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})
\]

\[
g(z) = \frac{1}{1 + e^{-z}} \quad \delta_i^{(l+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}}
\]
Backprop

\[ a^{(1)} = x \]
\[ z^{(2)} = \Theta^{(1)} a^{(1)} \]
\[ a^{(2)} = g \left( z^{(2)} \right) \]
\[ \vdots \]
\[ a^{(l)} = g \left( z^{(l)} \right) \]
\[ z^{(l+1)} = \Theta^{(l)} a^{(l)} \]
\[ a^{(l+1)} = g \left( z^{(l+1)} \right) \]
\[ \vdots \]
\[ \hat{y} = a^{(L+1)} \]

\[ \frac{\partial L(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)} \]

\[ \delta_i^{(l)} = a_i^{(l)} (1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)} \]

\[ \delta_i^{(L+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(L+1)}} = \frac{\partial}{\partial z_i^{(L+1)}} \left[ y \log(g(z^{(L+1)})) + (1 - y) \log(1 - g(z^{(L+1)})) \right] \]
\[ = \frac{y}{g(z^{(L+1)})} g'(z^{(L+1)}) - \frac{1 - y}{1 - g(z^{(L+1)})} g'(z^{(L+1)}) \]
\[ = y - g(z^{(L+1)}) = y - a^{(L+1)} \]

\[ L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) \]

\[ g(z) = \frac{1}{1 + e^{-z}} \quad \delta_i^{(l+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \]
Backprop

\[ a^{(1)} = x \]
\[ z^{(2)} = \Theta^{(1)} a^{(1)} \]
\[ a^{(2)} = g \left( z^{(2)} \right) \]
\[ \vdots \]
\[ a^{(l)} = g \left( z^{(l)} \right) \]
\[ z^{(l+1)} = \Theta^{(l)} a^{(l)} \]
\[ a^{(l+1)} = g \left( z^{(l+1)} \right) \]
\[ \vdots \]
\[ \hat{y} = a^{(L+1)} \]

\[ L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) \]

\[ g(z) = \frac{1}{1 + e^{-z}} \quad \delta^{(l+1)}_i = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \]

\[ \partial L(y, \hat{y}) \partial \Theta_{i,j}^{(l)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta^{(l+1)}_i \cdot a_j^{(l)} \]

\[ \delta^{(l)}_i = a_i^{(l)} (1 - a_i^{(l)}) \sum_k \delta^{(l+1)}_k \cdot \Theta_{k,i}^{(l)} \]

\[ \delta^{(L+1)} = y - a^{(L+1)} \]

Recursive Algorithm!
Auto-differentiation

Backprop for this simple network architecture is a special case of **reverse-mode auto-differentiation**:

\[
y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2)
\]

Forward Primal Trace:
- \( v_{-1} = x_1 = 2 \)
- \( v_0 = x_2 = 5 \)
- \( v_1 = \ln v_{-1} = \ln 2 \)
- \( v_2 = v_{-1} \times v_0 = 2 \times 5 \)
- \( v_3 = \sin v_0 = \sin 5 \)
- \( v_4 = v_1 + v_2 = 0.693 + 10 \)
- \( v_5 = v_4 - v_3 = 10.693 + 0.959 \)

Reverse Adjoint (Derivative) Trace:
- \( \bar{v}_1 = \bar{v}_{-1} = 5.5 \)
- \( \bar{x}_2 = \bar{v}_0 = 1.716 \)
- \( \bar{v}_{-1} = \bar{v}_{-1} + \bar{v}_1 \frac{\partial y}{\partial v_{-1}} = \bar{v}_{-1} + \bar{v}_1 / v_{-1} = 5.5 \)
- \( \bar{v}_0 = \bar{v}_0 + \bar{v}_2 \frac{\partial y}{\partial v_0} = \bar{v}_0 + \bar{v}_2 \times v_{-1} = 1.716 \)
- \( \bar{v}_1 = \bar{v}_2 \frac{\partial y}{\partial v_1} = \bar{v}_2 \times v_0 = 5 \)
- \( \bar{v}_2 = \bar{v}_3 \frac{\partial y}{\partial v_2} = \bar{v}_3 \times \cos v_0 = -0.284 \)
- \( \bar{v}_3 = \bar{v}_4 \frac{\partial y}{\partial v_3} = \bar{v}_4 \times 1 = 1 \)
- \( \bar{v}_4 = \bar{v}_5 \frac{\partial y}{\partial v_4} = \bar{v}_5 \times 1 = 1 \)
- \( \bar{v}_5 = \bar{y} = 1 \)
Auto-differentiation

• Given a function, computes its partial derivatives

• Compute all of the partial derivatives of a function with (nearly) same computation runtime [Griewank ’89, Baur and Strassen ’83]

• Backbone of (applied) machine learning: Pytorch, Tensorflow, …
Example of Computation Graph

$$f(w_1, w_2) = \left( \sin \left( \frac{2\pi w_1}{w_2} \right) + \frac{3w_1}{w_2} - \exp(2w_2) \right) \cdot \left( \frac{3w_1}{w_2} - \exp(2w_2) \right)$$

**Input:** $z_0 = (w_1, w_2)$

1. $z_1 = \frac{w_1}{w_2}$
2. $z_2 = \sin(2\pi \cdot z_1)$
3. $z_3 = \exp(2w_2)$
4. $z_4 = 3z_1 - z_3$
5. $z_5 = z_2 + z_4$
6. $z_6 = z_4 \cdot z_5$

**Return:** $z_6$
Computation Model

• Given access to a set of differentiable real functions $h \in \mathcal{H}$
• Use functions in $\mathcal{H}$ to create intermediate variables.
• Evaluation trace:
  • All intermediate variables will be scalars; each corresponds to a node.
  • Input $z_0 = w \in \mathbb{R}^d$. $[z_0]_1 = w_1, [z_0]_2 = w_2, \ldots, [z_0]_d = w_d$
  • Step 1: $z_1 = h_1$ (a subset of variables in $w$)
  • $\ldots$ $z_t = h_t$ (a subset of variables in $z_1, \ldots, z_{t-1}, w$)
  • $\ldots$
  • Step $T$: $z_T = h_T$ (a subset of variables in $z_1, \ldots, z_{T-1}, w$)
• Return: $z_T$

$(h_1, \ldots, h_T \in \mathcal{H})$
Computation Model

• Every $h \in \mathcal{H}$ is one of the following:
  • Type 1: An affine transformation of the inputs
    $$\beta z_i - z_j, \ z_i + z_j, \ z_i z_j, \ z_i^2 + z_j + 6$$
  • Type 2: A product of variables, to some power
    $$\frac{w_i}{w_j}, \ z_i, \ z_i^4, \ z_i^6 = z_i^4 z_j^2 z_6^{-1}$$
  • Type 3: A fixed set of one dimensional differentiable functions: $\sin(\cdot), \cos(\cdot), \exp(\cdot), \log(\cdot), \ldots$
    • We assume we can easily compute the derivatives for each of this functions.
    • Type 3 can be approximated by Type 1 and Type 2, using polynomials.
Neural Network Example

\[ \text{constant} \]

\[ a^{(1)} \rightarrow a^{(2)} \rightarrow a^{(3)} \rightarrow a^{(4)} \]
Reverse Mode of Automatic Differentiation

**Goal:** Compute partial derivatives of \( f(w) \), i.e., \( df/dw \).

- **Step 1:** computer \( f(w) \) and store in memory all intermediate variables \( z_1, \ldots, z_T \).

- **Step 2:** Initialize: \( \frac{dz_T}{dz_T} = 1 \).

- **Step 3:** For \( t = T, T-1, \ldots, 0 \)
  
  \[ \frac{dz_T}{dz_t} = \sum c \text{ is a child of } t \]

  \( \frac{d}{dz_c} \cdot \frac{d}{dz_t} \)

  (Child: a node \( z_t \) directly points to)

- **Step 4:** Return \( \frac{dz_T}{dz_0} = \frac{df}{dw} \)
Time Complexity

Theorem (Baur and Strassen ’83, Griewak ’89): Assume every $h$ is specified as in our computational model. For $h(\cdot)$ of type 3, assume we can compute the derivative $h'(z)$ in time as the same order of computing $h(z)$. Let $T$ denote the time to compute $f(w)$. Then the reverse mode computes $df/dw$ in time $O(T)$.

\[ \text{Type 1: } Z_c = a^T (z_1, \ldots, z_t) + b \rightarrow \text{Coefficients} \]
\[ \text{Type 2: } \text{Product: } \frac{\partial Z_c}{\partial z_t} = a \cdot \frac{\partial z_c}{\partial z_t}, \quad a: \text{exponent}, \quad z_c = z_1 \cdot z_t \]
\[ \text{Type 3: } Z_c = h(z_t), \quad \frac{\partial Z_c}{\partial z_t} = h'(z_t) \]

2) Time: $O(V+\tilde{E})$
Clarke Differential
Subdifferential and Subgradient

**Definition:** Given $f : \mathbb{R}^d \rightarrow \mathbb{R}$, for every $x$, the subdifferential set is defined as

$$\partial_s f(x) \triangleq \{ s \in \mathbb{R}^d : \forall x' \in \mathbb{R}^d, f(x') \geq f(x) + s^T (x' - x) \}.$$ The elements in the subdifferential set are subgradients.

$$\forall s \in \partial f(x)$$

$$x_{t+1} \leq x_t - \eta G_f$$
Subdifferential and Subgradient

**Definition:** Given $f : \mathbb{R}^d \to \mathbb{R}$, for every $x$, the subdifferential set is defined as

$$\partial_s f(x) \triangleq \{ s \in \mathbb{R}^d : \forall x' \in \mathbb{R}^d, f(x') \geq f(x) + s^T(x' - x) \}.$$  

The elements in the subdifferential set are subgradients.

- If $f$ is convex → $\partial_s f$ exists everywhere
- If $f$ is convex & differentiable
  $$\partial_s f(x) = \{ \nabla f(x) \}$$
- $O \left( \frac{1}{\sqrt{T}} \right)$ rate
Subdifferential is not enough

**Definition:** Given $f : \mathbb{R}^d \to \mathbb{R}$, for every $x$, the subdifferential set is defined as

$$\partial_s f(x) \triangleq \{ s \in \mathbb{R}^d : \forall x' \in \mathbb{R}^d, f(x') \geq f(x) + s^\top (x' - x) \}.$$  

The elements in the subdifferential set are subgradients.

**Problem:** $NN$ is not convex

![Graph showing a non-convex function and subgradient issue]

Subgradient is not well-defined

**Example:**
- Choose $x = -2$
- Choose $x' = 1$

We need $s$ such that

$$f(x') \geq (1 + s \cdot (x' - (-1)))$$

- Choose $x = -1$
- Choose $x' = 1$

$$0 \geq 1 + s \cdot (2 - (-1))$$

$$\Rightarrow s \geq 1$$

$$1 \geq 1 + s \cdot 2 \Rightarrow s \leq 0$$
Clarke Differential

**Definition:** Given $f : \mathbb{R}^d \rightarrow \mathbb{R}$, for every $x$, the Clark differential is defined as

$$
\partial f(x) = \text{conv} \left( \left\{ s \in \mathbb{R}^d : \exists \{x_i\}_{i=1}^{\infty} \rightarrow x, \{\nabla f(x_i)\}_{i=1}^{\infty} \rightarrow s \right\} \right).
$$

The elements in the subdifferential set are subgradients.
When does Clarke differential exists

Definition (Locally Lipschitz): \( f : \mathbb{R}^d \rightarrow \mathbb{R} \) is locally Lipschitz if \( \forall x \in \mathbb{R}^d \), there exists a neighborhood \( S \) of \( x \), such that \( f \) is Lipschitz in \( S \).

\[ \forall x, x' \in S \, \| f(x) - f(x') \| \leq C \cdot \| x - x' \| \]

- It locally Lip \( \implies \) \( \exists f \) exist everywhere

- If \( f \) is convex \( \implies \) \( \partial f = \partial sf \)

- If \( f \) is differentiable \( \implies \) \( \partial f \{ f(x) \} \)

\( \star \) satisfies chain rule