Gradient Descent Optimization

- Definition
- Mathematical calculation of gradient
- Matrix interpretation of gradient computation
In order to train, we need to minimize loss

$$\theta^* = \arg \min_{\theta} \{L(\theta)\}$$

- How do we do this?

Key ideas:
- Use gradient descent
- Computing gradient using chain rule, adjoint gradient, back propagation.
What is Gradient Descent?

- **Gradient descent:**
  - The simplest (but surprisingly effective) approach
  - Move directly down hill

- **What is the down hill direction?**
  \[ d = -\nabla L(\theta) \]

- **Gradient descent algorithm**

```plaintext
Repeat until converged {
    \[ d \leftarrow -\nabla L(\theta) \]
    \[ \theta \leftarrow \theta + \alpha d^T \]
}
```

*Gradient Descent (GD) Algorithm*
The GD update step:

\[ d \leftarrow -\nabla L(\theta) \]
\[ \theta \leftarrow \theta + \alpha d^t \]

1D case

Gradient Descent

starting point

ending point

2D case

Function

Updates
**Gradient Step Size**

- **How large should \( \alpha \) be?**
  - \( \alpha \) too small \( \Rightarrow \) slow convergence
  - \( \alpha \) too large \( \Rightarrow \) unstable
  - Often there is no good choice!

Too small \( \Rightarrow \) slow convergence

Too large \( \Rightarrow \) oscilate

Goldilocks? \( \Rightarrow \) good enough
Steepest Descent

- Use **line search** to compute the best $\alpha$

Repeat until converged {
  
  $d \leftarrow -\nabla L(\theta)$
  
  $\alpha^* \leftarrow \arg \min_{\alpha} \{L(\theta + \alpha d^t)\}$
  
  $\theta \leftarrow \theta + \alpha^* d^t$

}  

*Steepest Descent Algorithm*
Coordinate Descent

- Update one parameter at a time
  - Removes problem of selecting step size
  - Each update can be very fast, but lots of updates
Slow Convergence of Gradient Descent

- Very sensitive to **condition number** of problem
  - No good choice of step size

- Newton’s method: Correct for local second derivative
  - “Sphere” the ellipse
  - Too much computation; Too difficult to implement

- Alternative methods
  - Preconditioning: Easy, but tends to be ad-hoc, not so robust
  - Momentum: More latter

\[ \frac{\sigma_{\text{max}}}{\sigma_{\text{min}}} = 2 \]

\[ \frac{\sigma_{\text{max}}}{\sigma_{\text{min}}} = 10 \]
Computing the Loss Gradient

- Use chain rule to compute the loss gradient

\[ \nabla_\theta L_{MSE}(\theta) = \nabla_\theta \left\{ \frac{1}{K} \sum_{k=0}^{K-1} \| x_k - f_\theta(y_k) \|^2 \right\} \]

\[ = \frac{1}{K} \sum_{k=0}^{K-1} \nabla_\theta \{ \| x_k - f_\theta(y_k) \|^2 \} \]

\[ = \frac{2}{K} \sum_{k=0}^{K-1} (x_k - f_\theta(y_k))^t \nabla_\theta (x_k - f_\theta(y_k)) \]

\[ = -\frac{2}{K} \sum_{k=0}^{N-1} (x_k - f_\theta(y_k))^t \nabla_\theta f_\theta(y_k) \]

What does this mean?
**Interpretation of Loss Gradient**

Loss Gradient computation requires:

- Sum over training data: Big sum, but straightforward.
- Prediction error: Easy to compute.
- Gradient of inference function: This is the difficult part.
  - Most challenging part to compute.
  - Enabled by automatic differentiation built into modern domain specific languages (DSL) such as Pytorch, Tensorflow, and others.
  - For NN this is known as back propagation.

\[
-\nabla_\theta L_{\text{MSE}}(\theta) = \frac{2}{K} \sum_{k=0}^{N-1} (x_k - f_\theta(y_k)) \cdot \nabla_\theta f_\theta(y_k)
\]
Matrix Interpretation

- Since \( x_k = f_\theta(y_k) + \text{error} \), we have that

\[
\begin{align*}
1 \times N_x & \quad \text{error vector} \\
\epsilon_k^t &= (x_k - f_\theta(y_k))^t = \text{error vector}
\end{align*}
\]

- Then the parameter vector is given by

\[
\begin{align*}
P \times 1 & \quad \text{parameter vector} \\
\grad_\theta L_{MSE}^t &= \text{dimension of parameter vector}
\end{align*}
\]
Inference function gradient, $\nabla f_\theta(y_n)$, is given by

$$\frac{\partial [f_\theta(y_k)]_i}{\partial \theta_j}$$

$N_x \times P$ function gradient

$= \nabla_\theta f_\theta(y_k) = \text{gradient of function}$
Forward vs Backward Propagation

- **Forward gradient**

  \[ \delta \hat{x} \leftarrow \nabla_{\theta} f_{\theta}(y) \delta \theta \]

- **Backward (adjoint) gradient**

  \[ \epsilon^t \nabla_{\theta} f_{\theta}(y) \to d \]
Loss Gradient Computation

- **Equation is**

\[-\nabla_{\theta}L_{MSE}(\theta) = \frac{2}{K} \sum_{k=0}^{N-1} (x_k - f_\theta(y_k))^t \nabla_\theta f_\theta(y_k)\]

- **Looks like**

\[d_{1\times P} = \frac{2}{K} \sum_{k=0}^{K-1} \epsilon_k_{1\times N_x} \nabla_\theta f_\theta(y_k)\]
Update Direction for Supervised Training

- \( d \) is given by

\[
\begin{align*}
\mathbf{d} &_{1 \times P} = 2 \sum_{k=0}^{K-1} \frac{1}{K} \mathbf{\epsilon}_k \mathbf{\epsilon}_k^T \mathbf{\nabla}_\theta f_\theta(y_k) \\
\mathbf{\epsilon}_k &_{1 \times N_x} \mathbf{\nabla}_\theta f_\theta(y_k) &_{N_x \times P}
\end{align*}
\]