Optimization of Deep Functions

- Intro to Deep Learning
- Deep Learning Structures
- Gradient of a Deep Network
- Forward and Back Propagation
Deep Learning Background

- **Shallow learning:**
  - Inference function formed by 1 or 2 layers.
  - For many years, this was thought to be enough [Cybenko 1989].

- **Deep learning:**
  - What is it?
    - Uses many layers of inference functions in a hierarchal structure.
    - Typically 10 to 100 layers.
  - Why do people use it?
    - Over the last decade, there has been overwhelming empirical evidence that deep learning dramatically outperforms shallow learning on a wide range of real applications.
    - Intuitively, it is easier to represent complex phenomena using a hierarchy of inference functions.
  - If it’s so great, then why did it take so long?
    - It was difficult to implement and no one was sure it would work, but now it's easy to implement with modern software tools.
    - Must train all layers jointly ⇒ requires automatic differentiation
    - Many more parameters to model and the problem of “vanishing gradient”
    - Requires more training data.
- Inference function is hierarchical.
  - For $M = 4$, we have that $\theta = [\theta_0, \theta_1, \theta_2, \theta_3]$, and

$$\hat{x} = f_\theta(y) = f_{3,\theta_3} \left( f_{2,\theta_2} \left( f_{1,\theta_1} \left( f_{0,\theta_0}(y) \right) \right) \right)$$

  (this is a mess)

  - Instead write

$$f_\theta y = f_{3,\theta_3} f_{2,\theta_2} f_{1,\theta_1} f_{0,\theta_0} y = \left[ \prod_{m=0}^{M-1} f_{m,\theta_m} \right] y$$
Remember the Loss Gradient

- To compute the loss gradient

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

- For each training pair, we need to:
  - Compute the error vector
  - Multiply by the adjoint gradient $\Leftarrow$ This is the hard part!
  - Sum them up
The Adjoint of the Loss Gradient

- To compute the loss gradient

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

- The adjoint gradient of the loss function:
  - Input and output are exchanged
  - Reverses flow of network
Example Deep Network

- For $M = 3$,
  - Inference function $f_{\theta}y = f_{2,\theta_2}f_{1,\theta_1}f_{0,\theta_0}y$
  - Parameter vectors is $\theta = [\theta_0, \theta_1, \theta_2]$
  - Hidden states $z_{m+1} = f_{\theta_m}z_m$ with dimension $z_m \in \mathbb{R}^{N_m}$

- How do we compute the gradient of this inference function?
By the chain rule:

- Gradient with respect to $z_0$
  \[
  \nabla_{z_0} f_{\theta} = [\nabla_{z_2} f] [\nabla_{z_1} f] [\nabla_{z_0} f]
  \]

- Adjoint gradient with respect to $z_0$
  \[
  [\nabla_{z_0} f_{\theta}]^t = [\nabla_{z_0} f] [\nabla_{z_1} f] [\nabla_{z_2} f]^t
  \]

- Gradient with respect to $\theta_0$
  \[
  \nabla_{\theta_0} f_{\theta} = [\nabla_{z_2} f] [\nabla_{z_1} f] [\nabla_{\theta_0} f]
  \]

- Gradient with respect to $\theta_0$
  \[
  [\nabla_{\theta_0} f_{\theta}]^t = [\nabla_{\theta_0} f] [\nabla_{z_1} f] [\nabla_{z_2} f]^t
  \]
Gradients for Each Node

- **Input gradient of layer:**
  \[ A_m \leftarrow \nabla_z f_{\theta_m}(z_m) = \frac{\partial [f_{\theta_m}]_i}{\partial [z_m]_j} \]

- **Parameter gradient of layer:**
  \[ B_m \leftarrow \nabla_\theta f_{\theta_m}(z_m) = \frac{\partial [f_{\theta_m}]_i}{\partial [\theta_m]_j} \]
Adjoint Gradients for Each Node

• Input gradient of layer:

\[
A^t_m \leftarrow \left[ \nabla_z f_{\theta_m}(z_m) \right]^t = N_m \times N_{m+1} \text{ input gradient}
\]

\[
\frac{\partial [f_{\theta_m}]_j}{\partial [z_m]_i}
\]

• Parameter gradient of layer:

\[
B^t_m \leftarrow \left[ \nabla_{\theta} f_{\theta_m}(z_m) \right]^t = P_m \times N_{m+1} \text{ parameter gradient}
\]

\[
\frac{\partial [f_{\theta_m}]_j}{\partial [\theta_m]_i}
\]
By the chain rule, the gradients with respect to inputs are
\[ \nabla_{z_2} f_{\theta} = A_2 \]
\[ \nabla_{z_1} f_{\theta} = A_2 A_1 \]
\[ \nabla_{z_0} f_{\theta} = A_2 A_1 A_0 \]

By the chain rule, the gradients with respect to parameters are
\[ \nabla_{\theta_2} f_{\theta} = B_2 \]
\[ \nabla_{\theta_1} f_{\theta} = A_2 B_1 \]
\[ \nabla_{\theta_0} f_{\theta} = A_2 A_1 B_0 \]
Back Propagation of Adjoint Gradient

- Compute loss gradient by reversing network flow:
  - For each training vector,
    \[ g_{m,k} = \nabla_{\theta_m} \| x_k - f_\theta(y_k) \|^2 \]
  - So then the gradient of the loss function is given by
    \[ \nabla_{\theta_m} L(\theta) = g_m = \frac{1}{K} \sum_{k=0}^{K-1} g_{m,k} \]
BP Step 1: Forward Propagate States

- Forward propagation pseudo code:

\[
\begin{align*}
\hat{x} & \leftarrow z_M \\
& \text{For } (m = 0 \text{ to } M - 1) \{ \\
& \quad z_{m+1} \leftarrow f_{\theta_m} z_m \\
& \} \\
& z_0 \leftarrow y
\end{align*}
\]

Forward Propagation

M=3 Layer Network
**BP Step 2: Back Propagation**

*Backpropagation pseudo code*

For\((m = 0 \text{ to } M - 1)\) \(g_m \leftarrow 0\)

For \((k = 0 \text{ to } K - 1)\) {
\[\epsilon_M \leftarrow -2(x_k - f_\theta(y_k))\]
For \((m = M - 1 \text{ to } 0)\) {
\[A \leftarrow \nabla_z f_\theta_m(z_m)\]
\[B \leftarrow \nabla_\theta f_\theta_m(z_m)\]
\[\epsilon_m \leftarrow A^t \epsilon_{m+1}\]
\[g_m \leftarrow g_m + B^t \epsilon_{m+1}\]
}
For \((m = 0 \text{ to } M - 1)\) \(g_m \leftarrow g_m/K\)

Return \((g_m)\)

\[\text{Results in}\]
\[g_m = \nabla_\theta_m L(\theta)\]

where
\[L(\theta) = \frac{1}{K} \sum_{k=0}^{K-1} \|x_k - f_\theta(y_k)\|^2\]
Back Propogation: Computational Efficiency

- **Order of computation matters!!**
  - Left to Right: Dense matrix products $\Rightarrow O(N^3K)$
  - Right to Left: Vector matrix products $\Rightarrow O(N^2K)$

\[ g_o = \frac{1}{K} \sum_{k=0}^{K-1} \begin{pmatrix} P_0 \times N_1 \\
\text{Loss gradient} \\
\end{pmatrix} + \begin{pmatrix} N_1 \times N_2 \\
\text{input gradient} \\
\end{pmatrix} + \begin{pmatrix} N_2 \times N_3 \\
\text{input gradient} \\
\end{pmatrix} \]

This fast algorithm is sometimes referred to as adjoint differentiation.
GD on Acyclic Graph Structures

- Definition of DAG
- General Node Structure
- Back Propagation on Graphs
Define DAG

- Directed Acyclic Graph (DAG)
  - A directed graph with no loops.
  - Starting at any node, you will eventually hit a termination

  ![Directed Acyclic Graph (DAG)](image)

  - If you reverse the arrows (i.e., flow), it’s still a DAG

  ![Reversed Directed Acyclic Graph](image)
Define DL Node

- Assume that DL network is a DAG.
  - Then w.o.l.o.g for a fan-in of 2, each node is

\[
x = f_{s,\theta}(y_1, y_2)
\]

\[
A_{s,1} = \nabla_{y_1} f_s
\]

\[
A_{s,2} = \nabla_{y_2} f_s
\]

\[
B_s = \nabla_{\theta} f_s
\]

- And the back propagation is

\[
\delta_1
\]

\[
\delta_2
\]

\[
[\nabla f_s]^t
\]

\[
\epsilon
\]

\[
\theta
\]

Adjoint gradient propagation

More detailed view of adjoint gradient propagation
Step 1: Forward Propagation for DAG

- Forward propagation.

\[
\begin{bmatrix}
x_{k,1} \\
x_{k,2}
\end{bmatrix} = f_\theta \left(\begin{bmatrix}
y_{k,1} \\
y_{k,2}
\end{bmatrix}\right)
\]
Step 2: Back Propagation for DAG

- Back propagation using adjoint gradients.

\[
\begin{align*}
\epsilon_{k,1} &= -2(x_k - f_\theta(y_k)) \\
\epsilon_{k,2} &= -2(x_k - f_\theta(y_k)) \\
\end{align*}
\]

Gradient is given by

\[
g_s = \frac{1}{K} \sum_{k=0}^{K-1} g_{k,s}
\]
Implementation of Each Node

- For each node you need two functions:
  - Function for forward propagation
    \[ x \leftarrow f(z_1, z_2, \theta) \]
    You need a software implementation of this forward function.

  - Function for adjoint gradient (i.e., back propagation)
    \[ [\delta_1, \delta_2, g] \leftarrow G(\epsilon, z_1, z_2, \theta) \]
    You need a software implementation of this function that multiplies \( \epsilon \) by the adjoint gradient.

Very important: You don’t need to ever compute \([\nabla f_s]^t\)!!
General Loss Functions

- The Weakness of MSE Loss Functions
- MAE and Cross Correlation Loss Functions
- Back Propagating General Loss Functions
Weakness of MSE Loss

- MSE loss is given by
  \[
  L_{MSE} (\theta) = \frac{1}{K} \sum_{k=0}^{K-1} \| x_k - f_\theta(y_k) \|^2
  \]

- Excessively weights outliers
  - If training sample is wrong, then prediction error can be very large.
  - If a few bad training samples can dramatically degrade results.

\[
[\nabla L_{MSE} (\theta)]^t = \frac{-2}{K} \sum_{k=0}^{K-1} [\nabla f_\theta(y_k)]^t (x_k - f_\theta(y_k))
\]
A General Class of Loss Functions

- General loss functions

\[ L(\theta) = \frac{1}{K} \sum_{k=0}^{K-1} \rho(x_k, f_\theta(y_k)) \]

- where \( \rho \) is a distortion function s.t. \( \forall a, b, \rho(a, b) \geq \rho(a, a) \).

- Typical choices include

  - Mean Squared Error (MSE)
    \[ \rho(a, b) = ||a - b||^2 = \sum_i (a_i - b_i)^2 \quad \text{for} \ \forall a, b \in \mathbb{R}^{N_x} \]

  - Mean Absolute Error (MAE)
    \[ \rho(a, b) = ||a - b||_1^1 = \sum_i |a_i - b_i| \quad \text{for} \ \forall a, b \in \mathbb{R}^{N_x} \]

  - Multicategory Cross Entropy (assumes \( \sum_i b_i = 1 \))
    \[ \rho(a, b) = -\sum_i a_i \log b_i \quad \text{for} \ \forall a, b \in \mathbb{S}^{N_x} \]
Gradient for General for Loss

- Alternative loss functions

\[ L(\theta) = \frac{1}{K} \sum_{k=0}^{K-1} \rho(x_k, f_\theta(y_k)) \]

Results in gradient

\[ [\nabla L(\theta)]^t = \frac{1}{K} \sum_{k=0}^{K-1} [\nabla f_\theta(y_k)]^t \epsilon_k \]

\[ \epsilon_k = \nabla_b \rho(a, b) \bigg|_{a=x_k, b=f_\theta(y_k)} \]
**Error Vector for General Loss Functions**

\[ L(\theta) = \frac{1}{K} \sum_{k=0}^{K-1} \rho(x_k, f_\theta(y_k)) \]

General loss functions

- **MSE**
  
  Loss: \( \rho(a, b) = \|a - b\|^2 \)
  
  Error: \( \epsilon_k = \nabla_b \rho(a, b) \bigg|_{a=x_k, b=f_\theta(y_k)} \)

- **MAE**
  
  Loss: \( \rho(a, b) = \|a - b\|_1 \)
  
  Error: \( \epsilon_k^t = -\text{sign} \left( x_k - f_\theta(y_k) \right) \)

- **Cross entropy**\(^*\)
  
  Loss: \( \rho(a, b) = -\sum_i a_i \log b_i \)  
  
  (Assumes \( 1 = \sum_i b_i \) and \( a_i \geq 0 \))
  
  Error: \( \epsilon_k^t = -\frac{x_k}{f_\theta(y_k)} \)

*The software utility includes softmax preprocessing: https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html*
Gradient for Cross Entropy Loss

- Paradox:
  - Let $f_\theta(y) = \theta$ for $\sum_i \theta_i = 1$, then for cross-entropy we have that:
    \[
    L(\theta) = -\sum_i x_i \log \theta_i \quad \text{for} \quad \sum_i \theta_i = 1
    \]
    So that
    \[
    [\nabla_\theta L(\theta)]_i = [\nabla_\theta \rho(x, \theta)]_i = -\frac{x_i}{\theta_i}
    \]
  - Paradox: Even when $\hat{\theta}_i = \frac{x_i}{\sum_i x_i}$, the gradient is not zero!
    \[
    d = -\nabla_\theta L(\hat{\theta}) = [1,1,1]
    \]
  - How can this be??

3D Example:
Gradient is perpendicular to constraint space!
Back Propagation for Typical Loss Functions

- Loss gradient given by

\[
\nabla_{\theta_m} L(\theta) = \frac{1}{K} \sum_{k=0}^{K-1} g_{m,k}
\]

- General: \( \epsilon_k = \nabla_b \rho(a, b) |_{a=x_k, b=f_\theta(y_k)} \)
- MSE: \( \epsilon_k^t = -2(x_k - f_\theta(y_k)) \)
- MAE: \( \epsilon_k^t = -\text{sign}(x_k - f_\theta(y_k)) \)
- CE: \( \epsilon_k^t = -\frac{x_k}{f_\theta(y_k)} \)
DAG Back Propagation for General Loss

- Back propagation using adjoint gradients.

\[ \epsilon_{k,1} = \nabla_b \rho_1(a, b) \big|_{a=x_k}^{b=f_\theta(\gamma_k)} \]

\[ \epsilon_{k,2} = \nabla_b \rho_2(a, b) \big|_{a=x_k}^{b=f_\theta(\gamma_k)} \]

Gradient is given by

\[ g_s = \frac{1}{K} \sum_{k=0}^{K-1} g_{k,s} \]