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

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

(technically 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

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

- 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

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

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

- By the chain rule:
  - Gradient with respect to $z_0$
    \[ \nabla_{z_0} f_\theta = [\nabla_{z_2} f_2][\nabla_{z_1} f_1][\nabla_{z_0} f_0] \]
  - Adjoint gradient with respect to $z_0$
    \[ \left[ \nabla_{z_0} f_\theta \right]^t = \left[ \nabla_{z_2} f_0 \right]^t \left[ \nabla_{z_1} f_1 \right]^t \left[ \nabla_{z_2} f_1 \right]^t \]
  - Gradient with respect to $\theta_0$
    \[ \nabla_{\theta_0} f_\theta = [\nabla_{z_2} f_2][\nabla_{z_1} f_1][\nabla_{\theta_0} f_0] \]
  - Gradient with respect to $\theta_0$
    \[ \left[ \nabla_{\theta_0} f_\theta \right]^t = \left[ \nabla_{\theta_0} f_0 \right]^t \left[ \nabla_{z_1} f_1 \right]^t \left[ \nabla_{z_2} f_1 \right]^t \]
Gradients for Each Node

• Input gradient of layer:

\[ A_m \leftarrow \nabla_z f_{\theta_m}(z_m) = \nabla \left[ f_{\theta_m}(z_m) \right]_i \frac{\partial f_{\theta_m}}{\partial [z_m]_j} \]

• Parameter gradient of layer:

\[ B_m \leftarrow \nabla_\theta f_{\theta_m}(z_m) = \nabla \left[ f_{\theta_m}(z_m) \right]_i \frac{\partial f_{\theta_m}}{\partial [\theta_m]_j} \]
Adjoint Gradients for Each Node

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

- Parameter gradient of layer:
  \[ B_m^t \leftarrow [\nabla_\theta f_{\theta_m}(z_m)]^t = \]
  \[
  \frac{\partial [f_{\theta_m}]_j}{\partial [\theta_m]_i}
  \]
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,
    \[ g_{m,k} = \nabla_{\theta_m} \| y_k - f_\theta(x_k) \|^2 \]
  - 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:

\[
\begin{align*}
    z_0 &\leftarrow y \\
    \text{For } (m = 0 \text{ to } M - 1) \{ \\
    &\quad z_{m+1} \leftarrow f_{\theta_m}z_m \\
    \} \\
    \hat{x} &\leftarrow z_M
\end{align*}
\]
BP Step 2: Back Propagation

**Backpropagation pseudo code**

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

For \((k = 0\) to \(K - 1\)) {
    \[
    \epsilon_m \leftarrow -2(x_k - f_\theta(y_k))
    \]
    For \((m = M - 1\) 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 \((m = 0\) to \(M - 1\)) \(g_m \leftarrow g_m/K\)
Return \((g_m)\)
Back Propogation: Computational Efficiency

- Order of computation matters!!
  - Left to Right: Dense matrix products ⇒ $O(N^2K)$
  - Right to Left: Vector matrix products ⇒ $O(NK)$

This fast algorithm is sometimes referred to as adjoint differentiation.

$$g_o = \frac{1}{K} \sum_{k=0}^{K-1} \frac{P_0 \times N_1}{P_0} \text{ parameter gradient}$$

$$= \frac{1}{K} \sum_{k=0}^{K-1} \frac{N_1 \times N_2}{N_2} \text{ input gradient}$$

$$A_0^t \rightarrow A_1^t \rightarrow A_2^t \rightarrow -2(x_k - f_\theta(y_k)) \text{ error vector}$$
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

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

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

\[
[\nabla f_s]^t
\]

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.

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

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

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

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

- 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)
    \[ \rho(a, b) = \|a - b\|_1 = \sum_i |a_i - b_i| \quad \text{for } \forall a, b \in \mathbb{R}^{N_x} \]
  - Multicategory Cross Entropy
    \[ \rho(a, b) = -\sum_i a_i \log b_i \quad \text{for } \forall a, b \in \mathcal{S}^{N_x} \]
Gradient for General for Loss

- Alternative loss functions

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

Results in gradient

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

\[
\epsilon_k = \nabla_b \rho(a, b) \bigg|_{a=x_k}^{b=f_\theta(y_k)}
\]

- Loss gradient

\[
P \times 1
\]

- Function gradient

\[
P \times N_x
\]

- Error vector

\[
N_x \times 1
\]
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: \( \epsilon_k = -2(x_k - f_\theta(y_k)) \)

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

- **Cross entropy**
  
  Loss: \( \rho(a, b) = -\sum_i a_i \log b_i \)
  
  Error: \( \epsilon_k = -\frac{x_k}{f_\theta(y_k)} \)  
  
  Point-wise division
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: \( \epsilon_k = -2(x_k - f_\theta(y_k)) \)

MAE: \( \epsilon_k = -\text{sign}(x_k - f_\theta(y_k)) \)

CE: \( \epsilon_k = -\frac{x_k}{f_\theta(y_k)} \)
DAG Back Propagation for General Loss

- Back propagation using adjoint gradients.

Gradient is given by:

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