Table of Contents

1 Computational Graph

2 Functional Representation of DNN

3 Back Propagation
Table of Contents

1 Computational Graph

2 Functional Representation of DNN

3 Back Propagation
Consider the following function:

\[ f = \exp(\exp(x) + \exp(x)^2) + \sin(\exp(x) + \exp(x)^2) \]

The derivative of this function can be calculated easily.

A general algorithmic solution is to define some intermediate arguments:

\[
\begin{align*}
    a &= \exp(x) \\
    b &= a^2 \\
    c &= a + b \\
    d &= \exp(c) \\
    e &= \sin(c) \\
    f &= d + e
\end{align*}
\]
Recall the function:

\[ f = \exp(\exp(x) + \exp(x)^2) + \sin(\exp(x) + \exp(x)^2) \]

We can represent it in a computation graph.
Reverse Mode Auto Differentiation

- Computing the derivatives using the chain rule:

\[
\begin{align*}
\frac{df}{dd} &= 1 \\
\frac{df}{de} &= 1 \\
\frac{df}{dc} &= \frac{df}{dd} \frac{dd}{dc} + \frac{df}{de} \frac{de}{dc} \\
\frac{df}{db} &= \frac{df}{dd} \frac{dd}{db} + \frac{df}{de} \frac{de}{db} \\
\frac{df}{da} &= \frac{df}{dd} \frac{dd}{da} + \frac{df}{de} \frac{de}{da} \\
\frac{df}{dx} &= \frac{df}{dd} \frac{dd}{dx} + \frac{df}{de} \frac{de}{dx} \\
\frac{df}{dc} &= \frac{df}{dd} \exp(c) + \frac{df}{de} \cos(c) \\
\frac{df}{db} &= \frac{df}{dc} \exp(c) + \frac{df}{de} 2a \\
\frac{df}{da} &= \frac{df}{dc} \exp(c) + \frac{df}{de} \exp(x).
\end{align*}
\]

Source: Justin Domke, UMASS
# Table of Contents

1. Computational Graph
2. Functional Representation of DNN
3. Back Propagation
Consider a fully-connected NN for regression with the following architecture:
Expressing as a Composite Functions

- The input $x \in \mathbb{R}^n$

- 1st layer activations $\sigma_1(W_1 \cdot x + b_1)$ where $W_1 \in \mathbb{R}^{10 \times n}$ and $b_1 \in \mathbb{R}^{10}$.

- 2st layer activations $\sigma_2(W_2 \cdot \sigma_1(W_1 \cdot x + b_1) + b_2)$ where $W_2 \in \mathbb{R}^{5 \times 10}$, $b_2 \in \mathbb{R}^{5}$.

- 3rd layer activations

$$\hat{y} = W_3 \cdot \sigma_2(W_2 \cdot \sigma_1(W_1 \cdot x + b_1) + b_2) + b_3$$

where $W_3 \in \mathbb{R}^{1 \times 5}$, $b_3 \in \mathbb{R}$.

- The Empirical error (for squared loss)

$$\hat{L} = \frac{1}{2}(y - \hat{y})^2$$
Computing Gradients

- We would like to compute the gradients for every point on the training set in order to run GD
  
  \[
  \frac{\partial \hat{L}}{\partial W_1}, \frac{\partial \hat{L}}{\partial W_2}, \frac{\partial \hat{L}}{\partial W_3}, \frac{\partial \hat{L}}{\partial b_1}, \frac{\partial \hat{L}}{\partial b_2}, \frac{\partial \hat{L}}{\partial b_3}
  \]

- Computing all gradients can be computationally expensive

- Back-propagation will do it efficiently
Table of Contents

1 Computational Graph

2 Functional Representation of DNN

3 Back Propagation
We can write the architecture in a computation graph.
Back Propagation

\[
\frac{\partial \hat{L}}{\partial \hat{y}} = \frac{\partial \hat{L}}{\partial \hat{b}_3} = \frac{\partial \hat{L}}{\partial \hat{W}_3} = \frac{\partial \hat{L}}{\partial \hat{A}_2} =
\]
Back Propagation

\[
\frac{\partial \hat{L}}{\partial \hat{b}^2_2} = \frac{\partial \hat{L}}{\partial \hat{W}^2_2} = \frac{\partial \hat{L}}{\partial \hat{A}^1_1} = \frac{\partial \hat{L}}{\partial \hat{b}^1_1} = \frac{\partial \hat{L}}{\partial \hat{W}^1_1} =
\]
Regularization