# ELL 881: Fundamentals of Deep Learning

## Lec 03b: Deep Feedforward Networks:: Backpropagation

Vineet Kumar

August 10, 2018

# Table of Contents

# Table of Contents

# Forward Propagation

- MLP takes an input $\boldsymbol{x}$ and produces an output $\hat{\boldsymbol{y}}$
- **Information Flow**: Think of how information flows from $\boldsymbol{x}$ to generate $\hat{\boldsymbol{y}}$
- Forward Propagation: Information flows from the input layer to the next hidden layer, and finally to the output layer
- Thus, during training forward propagation continues until we compute the **loss** $J(\boldsymbol{\theta})$ function
- Back-Propagation or Backprop allows information from the loss to flow backward in order to compute gradients, and update the parameters.

# Backpropagation

- Backpropagation is only a method for computing gradients
- Question: How is a gradient related to derivative?
- Backpropagation is not a learning algorithm for MLP. Can you point some learning algorithms?
- Backpropagation is not limited to MLP. It is a **general method that can compute derivatives** of any function.
- More concretely, we will learn how to compute $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}, \boldsymbol{y})$ for a function $f$
- Here, $\boldsymbol{x}$ is the set of variables whose **derivates** we want to compute
- $\boldsymbol{y}$ is an additional set of variables to $f$
- For our task, we want to compute $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

# Table of Contents

# Computational Graphs

- **Node**: Indicates a variable
- **Operation**: Function of one or more variables
- In our discussion, we limit operations to return only a single output variable
- We add a directed edge $x \rightarrow y$ if an operation on $x$ yields $y$

# Computational Graphs: Example 1

a) $z = xy$, $\times$ is the multiplication operator

b) $\hat{y} = \sigma(x^T w + b)$

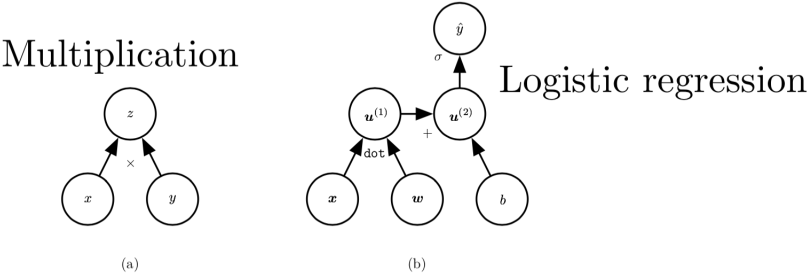


Figure 1: Computational Graphs

Image Courtesy: http://www.deeplearningbook.org/slides/06_mlp.pdf

# Computational Graphs: Example 2

c) **ReLU** $max\{0, XW + b\}$

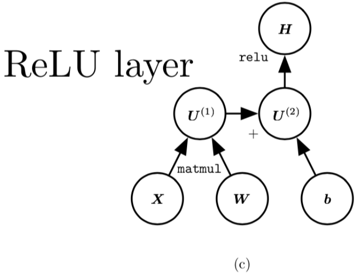


Figure 2: Computational Graph for ReLU
Image Courtesy: http://www.deeplearningbook.org/slides/06_mlp.pdf

# Table of Contents

# Chain Rule: Overview

- Chain rule is used to compute derivates of functions formed by composing other functions whose derivatives are known
- Back propagation uses chain rule, but in **a specific order** for efficiency.

# Chain Rule: Scalars

Let us assume $x$, $y$ and $z$ to be scalars

1. $y = g(x)$
2. $z = f(y)$

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

# Chain Rule: Vectors

Generalizing to vectors, let us say $\boldsymbol{x} \in \Re^m$, $\boldsymbol{y} \in \Re^n$

1. $\boldsymbol{y} = g(\boldsymbol{x})$
2. $z = f(\boldsymbol{y})$

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$

Gradient formulation:

$$\nabla_{\boldsymbol{x}} z = (\frac{\partial y}{\partial x})^T \nabla_{\boldsymbol{y}} z$$

Question: What are dimensions of $\frac{\partial y}{\partial x}$ ?
$\frac{\partial y}{\partial x}$ is $n \times m$, also known as **Jacobian matrix** of $g$

# Jacobian Gradient Product

$$\nabla_{\boldsymbol{x}} z = (\frac{\partial y}{\partial x})^T \nabla_{\boldsymbol{y}} z$$

- Jacobian-gradient product: Thus, gradient of a vector $\boldsymbol{x}$ can be obtained my multiplying Jacobian Matrix $\frac{\partial y}{\partial x}$ with a gradient $\nabla_{\boldsymbol{y}} z$
- Question: We saw gradient computation for a vector. How can this be generalized to a matrix or tensor?
  We can perhaps *flatten* a tensor to a vector, and then reshape the vector-gradient back to a tensor! Conceptually same as vector.

# Memory v/s Runtime tradeoff

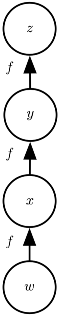


Figure 3: Figure 6.9
Image Courtesy:
http://www.deeplearningbook.org/slides/06_mlp.pdf

Let us assume we have a chain $x = f(w)$; $y = f(x)$ and $z = f(y)$

$$\frac{\partial z}{\partial w}$$

$$= \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w}$$

$$= f'(y)f'(x)f'(w)$$

Computationally efficient, used by Back-Prop!

$$= f'(f(f(w))f'(f(w))f'(w)$$

Memory friendly, we do not need to store $f(w)$!

# Symbol-to-Symbol Derivatives

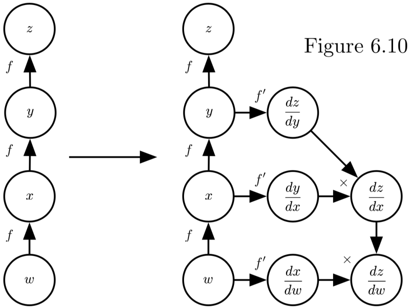


Figure 4:
Image Courtesy:
http://www.deeplearningbook.org/slides/06_mlp.pdf

- Tensorflow computes gradients by addition of nodes.
- You can see here that to compute $\frac{\partial z}{\partial w}$ you need to wait for nodes above it.