Automatic Differentiation

Guest lecture, DATA 558, Spring 2019
(Prof. Zaid Harchaoui instructor)

Vincent Roulet

Postdoc in Department of Statistics
University of Washington

UNIVERSITY *of* WASHINGTON

Machine Learning Pipeline

Machine Learning Pipeline

Machine learning pipeline

- Collect and preprocess data
  e.g. collect images, crop, center, normalize

Machine Learning Pipeline

Machine learning pipeline

- Collect and preprocess data

  e.g. collect images, crop, center, normalize
- Design model for given task

  e.g. linear classifier with logistic loss

Machine Learning Pipeline

Machine learning pipeline

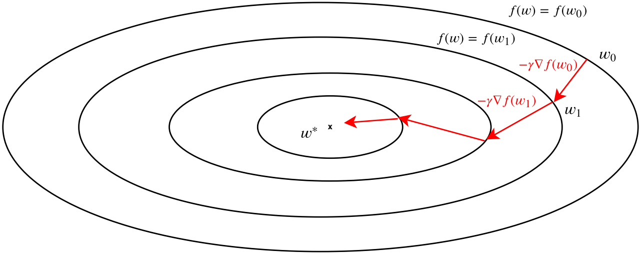
- Collect and preprocess data
  e.g. collect images, crop, center, normalize
- Design model for given task
  e.g. linear classifier with logistic loss
- Optimize your model,
  i.e., get parameters that minimize the error of the model
  e.g. using gradient descent on error f of your model

Machine Learning Pipeline

Machine learning pipeline

- Collect and preprocess data
  e.g. collect images, crop, center, normalize
- Design model for given task
  e.g. linear classifier with logistic loss
- Optimize your model,
  i.e., get parameters that minimize the error of the model
  e.g. using gradient descent on error f of your model

$$w \leftarrow w - \gamma \nabla f(w)$$



Machine Learning Pipeline

Machine learning pipeline

- Collect and preprocess data
  e.g. collect images, crop, center, normalize
- Design model for given task
  e.g. linear classifier with logistic loss
- Optimize your model,
  i.e., get parameters that minimize the error of the model
  e.g. using gradient descent on error of your model f

Bottleneck

- How to compute the gradients ?

Outline

Outline

Differentiation Methods

Simple Derivative Computation

Automatic Differentiation Toolbox

Gradient Computation

Gradient for Deep Neural Network

Advanced Derivatives

Differentiation Methods

Binary classification
Given sample $(x, y) \in \mathbb{R}^d \times \{-1, 1\}$ want to compute gradient of

$$f : w \to \log(1 + \exp(-y w^\top x))$$

Differentiation Methods

Binary classification
Given sample $(x, y) \in \mathbb{R}^d \times \{-1, 1\}$ want to compute gradient of

$$f : w \to \log(1 + \exp(-yw^\top x))$$

Solutions to compute the gradient:

1. Write down analytic form

$$\nabla f(w) = \frac{-yx}{1 + \exp(-yw^\top x)}$$

Differentiation Methods

Binary classification
Given sample $(x, y) \in \mathbb{R}^d \times \{-1, 1\}$ want to compute gradient of

$$f : w \to \log(1 + \exp(-yw^\top x))$$

Solutions to compute the gradient:

1. Write down analytic form

$$\nabla f(w) = \frac{-yx}{1 + \exp(-yw^\top x)}$$

Pros: Exact formulation, independent of the function evaluation
Cons: Need access to the analytic form of the function

Differentiation Methods

Binary classification
Given sample $(x, y) \in \mathbb{R}^d \times \{-1, 1\}$ wants to compute gradient of

$$f : w \to \log(1 + \exp(-yw^\top x))$$

Solutions to compute the gradient:

1. Write down analytic form
2. Use finite approximation

$$\nabla f(w)^\top d \approx \frac{f(w + \delta d) - f(w)}{\delta} \quad \text{for } 0 < \delta \ll 1$$

Differentiation Methods

Binary classification
Given sample $(x, y) \in \mathbb{R}^d \times \{-1, 1\}$ wants to compute gradient of

$$f : w \to \log(1 + \exp(-y w^\top x))$$

Solutions to compute the gradient:

1. Write down analytic form
2. Use finite approximation

$$\nabla f(w)^\top d \approx \frac{f(w + \delta d) - f(w)}{\delta} \quad \text{for } 0 < \delta \ll 1$$

Pros: Only needs access to the function evaluation of f
Cons: Inexact gradient

Differentiation Methods

Binary classification
Given sample $(x, y) \in \mathbb{R}^d \times \{-1, 1\}$ wants to compute gradient of

$$f : w \to \log(1 + \exp(-yw^\top x))$$

Solutions to compute the gradient:

1. Write down analytic form
2. Use finite approximation
3. Decompose f as successive compositions, use the chain-rule

Differentiation Methods

Binary classification
Given sample $(x, y) \in \mathbb{R}^d \times \{-1, 1\}$ wants to compute gradient of

$$f : w \to \log(1 + \exp(-y w^\top x))$$

Solutions to compute the gradient:

1. Write down analytic form
2. Use finite approximation
3. Decompose f as successive compositions, use the chain-rule

Automatic differentiation

Differentiation Methods

Binary classification
Given sample $(x, y) \in \mathbb{R}^d \times \{-1, 1\}$ wants to compute gradient of

$$f : w \to \log(1 + \exp(-yw^\top x))$$

Solutions to compute the gradient:

1. Write down analytic form
2. Use finite approximation
3. Decompose f as successive compositions, use the chain-rule

Automatic differentiation

Pros:
- Only needs access to the function evaluation by compositions
- Exact gradient

Outline

Simple Derivative Computation

Consider $\mathbb{R}^d = \mathbb{R}$, a sample $(x, y) = (3.5, 1)$, s.t.

$$f : w_0 \to \log(1 + \exp(-3.5 w_0))$$

Simple Derivative Computation

Consider $\mathbb{R}^d = \mathbb{R}$, a sample $(x, y) = (3.5, 1)$, s.t.

$$f : w_0 \to \log(1 + \exp(-3.5 w_0))$$

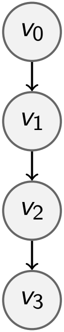
Function decomposition w_0 input, v_k successive evaluations

Simple Derivative Computation

Consider $\mathbb{R}^d = \mathbb{R}$, a sample $(x, y) = (3.5, 1)$, s.t.

$$f : w_0 \to \log(1 + \exp(-3.5w_0))$$

Function decomposition w_0 input, v_k successive evaluations



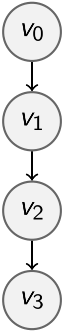
$$
\begin{aligned}
v_0 &= w_0 \\
v_1 &= -3.5v_0 \\
v_2 &= 1 + \exp(v_1) \\
v_3 &= \log(v_2)
\end{aligned}
$$

Simple Derivative Computation

Consider $\mathbb{R}^d = \mathbb{R}$, a sample $(x, y) = (3.5, 1)$, s.t.

$$f : w_0 \to \log(1 + \exp(-3.5 w_0))$$

Function decomposition w_0 input, v_k successive evaluations



$$v_0 = w_0$$

$$v_1 = -3.5 v_0$$

$$v_2 = 1 + \exp(v_1)$$

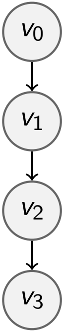
$$v_3 = \log(v_2) = f(w_0)$$

Simple Derivative Computation

Consider $\mathbb{R}^d = \mathbb{R}$, a sample $(x, y) = (3.5, 1)$, s.t.

$$f : w_0 \to \log(1 + \exp(-3.5 w_0))$$

Function decomposition w_0 input, v_k successive evaluations



$$
\begin{aligned}
v_0 &= w_0 \\
v_1 &= -3.5 v_0 \\
v_2 &= 1 + \exp(v_1) \\
v_3 &= \log(v_2) = f(w_0)
\end{aligned}
$$

$$f : w_0 \to g_2 \circ g_1 \circ g_0(w_0)$$

where
$$
\begin{aligned}
g_0 &: v_0 \to -3.5 v_0 \\
g_1 &: v_1 \to 1 + \exp(v_1) \\
g_2 &: v_2 \to \log(v_2)
\end{aligned}
$$

Chain Rule

Chain rule Given $f(w_0) = g_2 \circ g_1 \circ g_0(w_0)$,

$$f'(w_0) = g_0'(v_0)\, g_1'(v_1)\, g_2'(v_2)$$

where $v_0 = w_0, v_1 = g_0(v_0), v_2 = g_1(v_1)$

Chain Rule

Chain rule Given $f(w_0) = g_2 \circ g_1 \circ g_0(w_0)$,

$$f'(w_0) = g_0'(v_0)\, g_1'(v_1)\, g_2'(v_2)$$

where $v_0 = w_0, v_1 = g_0(v_0), v_2 = g_1(v_1)$

Only need derivatives of elementary functions

Chain Rule

Chain rule Given $f(w_0) = g_2 \circ g_1 \circ g_0(w_0)$,

$$f'(w_0) = g_0'(v_0)\, g_1'(v_1)\, g_2'(v_2)$$

where $v_0 = w_0, v_1 = g_0(v_0), v_2 = g_1(v_1)$

Only need derivatives of elementary functions

Elementary functions

- $v \to av$, $v \to v^k$, $v \to 1/v$
- $v \to \exp(v)$, $v \to \log(v)$, $v \to \cos(v)$, $v \to \sin(v)$
- ...

Forward-Backward Computation

Idea Recursive computations, using $\partial w_0 = \partial v_0$,

$$f'(w_0) = \frac{\partial f}{\partial v_0} = \frac{\partial v_1}{\partial v_0}\,\frac{\partial f}{\partial v_1} = \frac{\partial v_1}{\partial v_0}\,\frac{\partial v_2}{\partial v_1}\,\frac{\partial f}{\partial v_2} = \frac{\partial v_1}{\partial v_0}\,\frac{\partial v_2}{\partial v_1}\,\frac{\partial v_3}{\partial v_2}\,\frac{\partial f}{\partial v_3}$$

Forward-Backward Computation

Idea Recursive computations, using $\partial w_0 = \partial v_0$,

$$f'(w_0) = \frac{\partial f}{\partial v_0} = \frac{\partial v_1}{\partial v_0} \frac{\partial f}{\partial v_1} = \frac{\partial v_1}{\partial v_0} \frac{\partial v_2}{\partial v_1} \frac{\partial f}{\partial v_2} = \frac{\partial v_1}{\partial v_0} \frac{\partial v_2}{\partial v_1} \frac{\partial v_3}{\partial v_2} \frac{\partial f}{\partial v_3}$$

Algorithm

- Compute $\frac{\partial v_{k+1}}{\partial v_k} = g_k'(v_k)$ in a *forward* pass
- Compute $\frac{\partial f}{\partial v_k}$ in a *backward* pass using

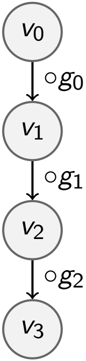
$$\frac{\partial f}{\partial v_k} = \frac{\partial v_{k+1}}{\partial v_k} \frac{\partial f}{\partial v_{k+1}}$$

Simple Derivative Computation

$$f(w_0) = \log(1 + \exp(-3.5w_0)), \qquad v_{k+1} = g_k(v_k) \qquad \lambda_k = \partial f / \partial v_k$$

Simple Derivative Computation

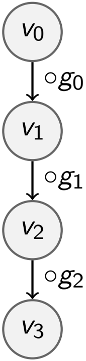
$$f(w_0) = \log(1 + \exp(-3.5 w_0)), \qquad v_{k+1} = g_k(v_k) \qquad \lambda_k = \partial f / \partial v_k$$



v_k
w_0
$-3.5 v_0$
$1 + \exp(v_1)$
$\log(v_2)$

Simple Derivative Computation

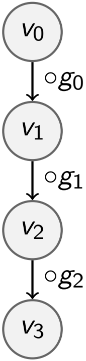
$$f(w_0) = \log(1 + \exp(-3.5 w_0)), \qquad v_{k+1} = g_k(v_k) \qquad \lambda_k = \partial f / \partial v_k$$



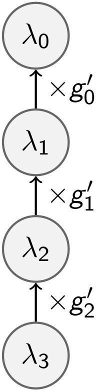
	v_k	$\partial v_k / \partial v_{k-1}$
v_0	w_0	1
v_1	$-3.5 v_0$	-3.5
v_2	$1 + \exp(v_1)$	$\exp(v_1)$
v_3	$\log(v_2)$	$1/v_2$

Simple Derivative Computation

$$f(w_0) = \log(1 + \exp(-3.5 w_0)), \qquad v_{k+1} = g_k(v_k) \qquad \lambda_k = \partial f / \partial v_k$$

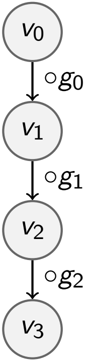


	v_k	$\partial v_k / \partial v_{k-1}$	$\partial f / \partial v_k$
v_0	w_0	1	
v_1	$-3.5 v_0$	-3.5	
v_2	$1 + \exp(v_1)$	$\exp(v_1)$	
v_3	$\log(v_2)$	$1/v_2$	



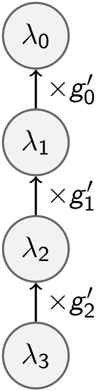
Simple Derivative Computation

$$f(w_0) = \log(1 + \exp(-3.5 w_0)), \qquad v_{k+1} = g_k(v_k) \qquad \lambda_k = \partial f / \partial v_k$$



	v_k	$\partial v_k / \partial v_{k-1}$
v_0	w_0	1
v_1	$-3.5 v_0$	-3.5
v_2	$1 + \exp(v_1)$	$\exp(v_1)$
v_3	$\log(v_2)$	$1/v_2$

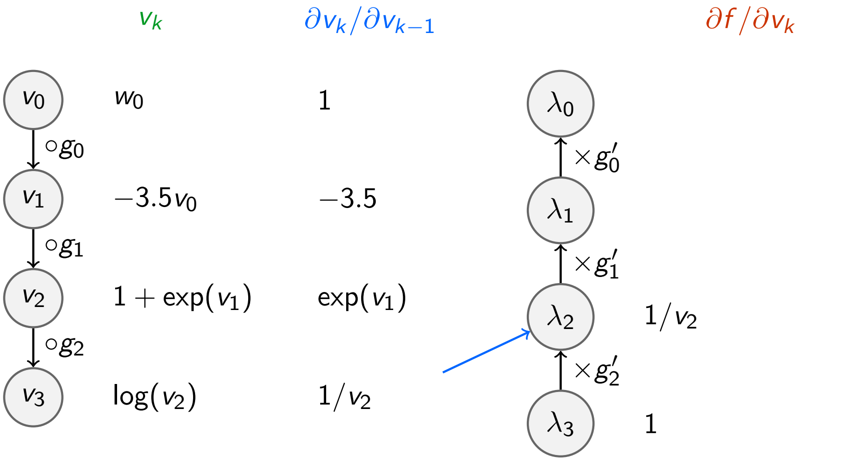
$\partial f / \partial v_k$



λ_3: 1

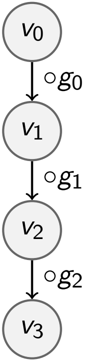
Simple Derivative Computation

$$f(w_0) = \log(1 + \exp(-3.5 w_0)), \qquad v_{k+1} = g_k(v_k) \qquad \lambda_k = \partial f / \partial v_k$$

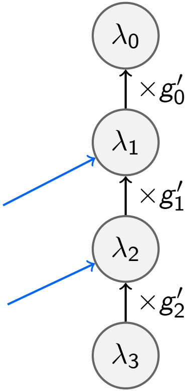


Simple Derivative Computation

$$f(w_0) = \log(1 + \exp(-3.5w_0)), \qquad v_{k+1} = g_k(v_k) \qquad \lambda_k = \partial f / \partial v_k$$



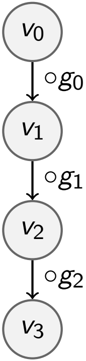
	v_k	$\partial v_k / \partial v_{k-1}$
v_0	w_0	1
v_1	$-3.5v_0$	-3.5
v_2	$1 + \exp(v_1)$	$\exp(v_1)$
v_3	$\log(v_2)$	$1/v_2$



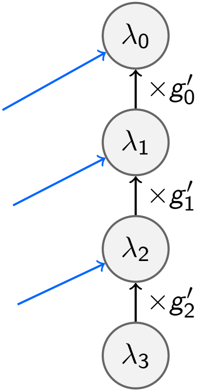
	$\partial f / \partial v_k$
λ_0	
λ_1	$\exp(v_1)/v_2$
λ_2	$1/v_2$
λ_3	1

Simple Derivative Computation

$$f(w_0) = \log(1 + \exp(-3.5 w_0)), \qquad v_{k+1} = g_k(v_k) \qquad \lambda_k = \partial f / \partial v_k$$



	v_k	$\partial v_k / \partial v_{k-1}$	$\partial f / \partial v_k$
v_0	w_0	1	$-3.5 \exp(v_1)/v_2$
v_1	$-3.5 v_0$	-3.5	$\exp(v_1)/v_2$
v_2	$1 + \exp(v_1)$	$\exp(v_1)$	$1/v_2$
v_3	$\log(v_2)$	$1/v_2$	1



Forward-Backward Computation

Forward pass $\frac{\partial v_{k+1}}{\partial v_k}$

- Compute $v_1 = g_0(v_0), v_2 = g_1(v_1), v_3 = g_2(v_2)$
- Store $\frac{\partial v_1}{\partial v_0} = g_0'(v_0), \frac{\partial v_2}{\partial v_1} = g_1'(v_1), \frac{\partial v_3}{\partial v_2} = g_2'(v_2)$

Forward-Backward Computation

Forward pass $\frac{\partial v_{k+1}}{\partial v_k}$

- Compute $v_1 = g_0(v_0), v_2 = g_1(v_1), v_3 = g_2(v_2)$
- Store $\frac{\partial v_1}{\partial v_0} = g_0'(v_0), \frac{\partial v_2}{\partial v_1} = g_1'(v_1), \frac{\partial v_3}{\partial v_2} = g_2'(v_2)$

Backward pass $\frac{\partial f}{\partial v_k}$

- Initialize $\frac{\partial f}{\partial v_3} = 1$
- For $k = 2, \ldots 0$,
- Compute $\frac{\partial f}{\partial v_k} = \frac{\partial v_{k+1}}{\partial v_k} \frac{\partial f}{\partial v_{k+1}}$
- Output $f'(w_0) = \frac{\partial f}{\partial v_0}$

Outline

Automatic Differentiation Toolboxes

From theory to practice

- Good to know how automatic differentiation works

Automatic Differentiation Toolboxes

From theory to practice

- Good to know how automatic differentiation works
- Hard to implement for generic decomposition or device

Automatic Differentiation Toolboxes

From theory to practice

- Good to know how automatic differentiation works
- Hard to implement for generic decomposition or device
- Use publicly available libraries such as
  $\rightarrow$ *PyTorch*, *TensorFlow*, *Theano* ...

Automatic Differentiation Toolboxes

From theory to practice

- Good to know how automatic differentiation works
- Hard to implement for generic decomposition or device
- Use publicly available libraries such as
  $\rightarrow$ *PyTorch*, *TensorFlow*, *Theano* ...

Here brief introduction to PyTorch

Automatic Differentiation in PyTorch

```
from torch import rand, log, exp

# Instantiate variable
w0 = rand(1)

# Flag to record any computation involving w0
w0.requires_grad = True

# Compute logistic loss on (x,y) = (3.5, 1)
out = log(1+exp(-3.5*w0))

# Backpropagate gradients of any input involved in out
out.backward()

# Derivative is recorded in the variable
print(w0.grad)
```

Automatic Differentiation Toolboxes

Features

- Vast library of elementary vectorial functions
- Easy construction of complex models by stacking operations
- Implemented in GPUs
  $\rightarrow$ fast back-propagation of convolution operations

Automatic Differentiation Toolboxes

Features

- Vast library of elementary vectorial functions
- Easy construction of complex models by stacking operations
- Implemented in GPUs
  $\rightarrow$ fast back-propagation of convolution operations

Main message

Can compute derivatives of any computations

Outline

Gradient Computation

Same forward-backward algorithm, replaces scalar by vectors,

$$f(w_0) = \sum_{i=1}^{n} \log(1 + \exp(-y_i w_0^\top x_i)),\ w_0 \in \mathbb{R}^d,\ x_i \in \mathbb{R}^d,\ y_i \in \{-1, 1\}$$

Gradient Computation

Same forward-backward algorithm, replaces scalar by vectors,

$$f(w_0) = \sum_{i=1}^{n} \log(1 + \exp(-y_i w_0^\top x_i)),\ w_0 \in \mathbb{R}^d,\ x_i \in \mathbb{R}^d,\ y_i \in \{-1, 1\}$$

$$f(w_0) = g_3 \circ g_2 \circ g_1 \circ g_0(w_0)$$

where, denoting $X = (y_1 x_1, \ldots, y_n x_n)^\top$, $\mathbf{1}_n = (1, \ldots, 1)$,

$$v_1 = g_0(v_0) = -Xv_0 \qquad v_3 = g_2(v_2) = \log(v_2)$$

$$v_2 = g_1(v_1) = \mathbf{1}_n + \exp(v_1) \qquad v_4 = g_3(v_3) = \mathbf{1}_n^\top v_3$$

Gradient Computation

Chain rule

$$f(w_0) = g_3 \circ g_2 \circ g_1 \circ g_0(w_0)$$
$$\nabla f(w_0) = \nabla g_0(v_0) \nabla g_1(v_1) \nabla g_2(v_2) \nabla g_3(v_3)$$

where - g_2, g_1, g_0 are multivariate functions, e.g., $g_0 : \mathbb{R}^d \to \mathbb{R}^n$
- g_3 is real-valued, i.e,. $g_3 : \mathbb{R}^n \to \mathbb{R}$

Gradient Computation

Chain rule

$$f(w_0) = g_3 \circ g_2 \circ g_1 \circ g_0(w_0)$$
$$\nabla f(w_0) = \nabla g_0(v_0) \nabla g_1(v_1) \nabla g_2(v_2) \nabla g_3(v_3)$$

where - g_2, g_1, g_0 are multivariate functions, e.g., $g_0 : \mathbb{R}^d \rightarrow \mathbb{R}^n$
- g_3 is real-valued, i.e,. $g_3 : \mathbb{R}^n \rightarrow \mathbb{R}$

Consequence: $\nabla g_0(v_0), \nabla g_1(v_1), \nabla g_2(v_2)$ are now matrices, $\nabla g_3(v_3)$ is a vector

Gradient Computation

Chain rule

$$f(w_0) = g_3 \circ g_2 \circ g_1 \circ g_0(w_0)$$
$$\nabla f(w_0) = \nabla g_0(v_0) \nabla g_1(v_1) \nabla g_2(v_2) \nabla g_3(v_3)$$

where - g_2, g_1, g_0 are multivariate functions, e.g., $g_0 : \mathbb{R}^d \to \mathbb{R}^n$
- g_3 is real-valued, i.e,. $g_3 : \mathbb{R}^n \to \mathbb{R}$

Consequence: $\nabla g_0(v_0), \nabla g_1(v_1), \nabla g_2(v_2)$ are now matrices,
$\nabla g_3(v_3)$ is a vector

Backward pass $\nabla_{v_k} f$ (vectors)

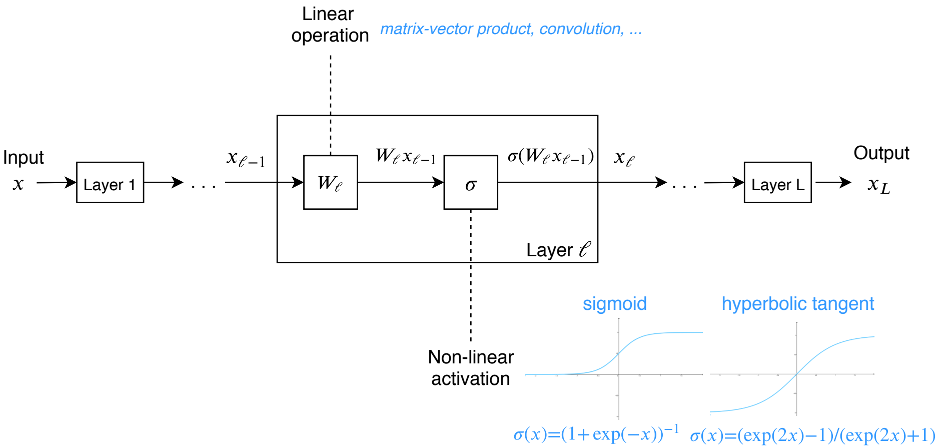
- Initialize $\nabla_{v2} f = \nabla g_3(v_3)$ (first step amounts to compute a vector)
- For $k = 1, \ldots 0$,
- Compute $\nabla_{v_k} f = \nabla_{v_k} v_{k+1} \nabla_{v_{k+1}} f$
  (iterations are matrix-vector products)
- Output $\nabla f(w_0) = \nabla_{v_0} f$

PyTorch Implementation

```
from torch import rand, log, exp

# Create data n =100, d = 20
X = rand(100, 20)

w0 = rand(20); w0.requires_grad=True

# Compute logistic loss
out = sum(log(1+ exp(-X.mv(w0))))

# Backpropagate gradients
out.backward()

print(w0.grad)
print(w0.grad.shape) # get 20 dimensional vector
```

Outline

Deep Neural Network



Deep Neural Network

Deep neural network structure

A deep neural network transforms an input $x = x_0$ using

$$x_\ell = \sigma_\ell(W_\ell \cdot x_{\ell-1}) \qquad \text{(Layer } \ell\text{)}$$

where σ_ℓ is the activation function, W_ℓ are the weights of the layer

Deep Neural Network

Deep neural network structure

A deep neural network transforms an input $x = x_0$ using

$$x_\ell = \sigma_\ell(W_\ell \cdot x_{\ell-1}) \qquad \text{(Layer } \ell\text{)}$$

where σ_ℓ is the activation function, W_ℓ are the weights of the layer

Objective

$$\min_{W=(W_0,\ldots,W_L)} \quad \frac{1}{n}\sum_{i=1}^{n} f^{(i)}(W) = \frac{1}{n}\sum_{i=1}^{n} f\left(y^{(i)}, x_L^{(i)}(W_0,\ldots,W_L)\right)$$

Deep Neural Network

Deep neural network structure

A deep neural network transforms an input $x = x_0$ using

$$x_\ell = \sigma_\ell(W_\ell \cdot x_{\ell-1}) \qquad \text{(Layer } \ell\text{)}$$

where σ_ℓ is the activation function, W_ℓ are the weights of the layer

Objective

$$\min_{W=(W_0,\dots,W_L)} \quad \frac{1}{n}\sum_{i=1}^{n} f^{(i)}(W) = \frac{1}{n}\sum_{i=1}^{n} f\left(y^{(i)}, x_L^{(i)}(W_0,\dots,W_L)\right)$$

with stochastic gradient descent

$$W \leftarrow W - \gamma \nabla f^{(i)}(W)$$

Gradient for Deep Neural Network

Binary classification with one hidden layer on $\mathbb{R}$
Given sample $(x, y) = (3.5, 1)$ wants to compute gradient of

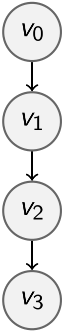
$$f : (w_0, w_1) \to (y - w_1 \tanh(xw_0))^2 = (1 - w_1 \tanh(3.5w_0))^2$$

Gradient for Deep Neural Network

Binary classification with one hidden layer on $\mathbb{R}$

Given sample $(x, y) = (3.5, 1)$ wants to compute gradient of

$$f : (w_0, w_1) \to (y - w_1 \tanh(xw_0))^2 = (1 - w_1 \tanh(3.5w_0))^2$$



$$v_0 = w_0$$

$$v_1 = \tanh(3.5v_0)$$

$$v_2 = w_1 v_1$$

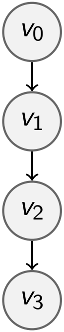
$$v_3 = (1 - v_2)^2$$

Gradient for Deep Neural Network

Binary classification with one hidden layer on $\mathbb{R}$

Given sample $(x, y) = (3.5, 1)$ wants to compute gradient of

$$f : (w_0, w_1) \to (y - w_1 \tanh(xw_0))^2 = (1 - w_1 \tanh(3.5w_0))^2$$



$$v_0 = w_0$$
$$v_1 = \tanh(3.5v_0)$$
$$v_2 = w_1 v_1$$
$$v_3 = (1 - v_2)^2$$

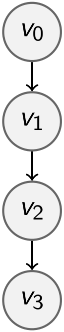
$$\frac{\partial f}{\partial w_0} = \frac{\partial f}{\partial v_1} \frac{\partial v_1}{\partial w_0}$$
$$\frac{\partial f}{\partial w_1} = \frac{\partial f}{\partial v_2} \frac{\partial v_2}{\partial w_1}$$

Gradient for Deep Neural Network

Binary classification with one hidden layer on $\mathbb{R}$

Given sample $(x, y) = (3.5, 1)$ wants to compute gradient of

$$f : (w_0, w_1) \to (y - w_1 \tanh(x w_0))^2 = (1 - w_1 \tanh(3.5 w_0))^2$$



$$v_0 = w_0$$

$$v_1 = \tanh(3.5 v_0)$$

$$v_2 = w_1 v_1$$

$$v_3 = (1 - v_2)^2$$

$$\frac{\partial f}{\partial w_0} = \frac{\partial f}{\partial v_1} \frac{\partial v_1}{\partial w_0}$$

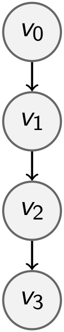
$$\frac{\partial f}{\partial w_1} = \frac{\partial f}{\partial v_2} \frac{\partial v_2}{\partial w_1}$$

$\to$ Use same computations $\frac{\partial f}{\partial v_\ell}$ for all layers

Gradient for Deep Neural Network

Binary classification with one hidden layer on $\mathbb{R}$
Given sample $(x, y) = (3.5, 1)$ wants to compute gradient of

$$f : (w_0, w_1) \to (y - w_1 \tanh(x w_0))^2 = (1 - w_1 \tanh(3.5 w_0))^2$$



$$v_0 = w_0$$
$$v_1 = \tanh(3.5 v_0)$$
$$v_2 = w_1 v_1$$
$$v_3 = (1 - v_2)^2$$

$$\frac{\partial f}{\partial w_0} = \frac{\partial f}{\partial v_1} \frac{\partial v_1}{\partial w_0}$$
$$\frac{\partial f}{\partial w_1} = \frac{\partial f}{\partial v_2} \frac{\partial v_2}{\partial w_1}$$

→ Use same computations $\frac{\partial f}{\partial v_\ell}$ for all layers
→ At layer ℓ, output

$$\frac{\partial f}{\partial w_\ell} = \frac{\partial f}{\partial v_\ell} \frac{\partial v_\ell}{\partial w_\ell}$$

PyTorch Implementation

```
from torch import rand, tanh

# Instantiate weights
w0 = rand(1); w1 = rand(1)

# Flag to record any computation involving w0 or w1
w0.requires_grad = True; w1.requires_grad = True

# Compute square loss of 1-layer DNN with tanh
    activation on (x,y)=(3.5,1)
out = tanh(3.5*w0)
out = w1*out
out = (1 - out)**2

# Backpropagate gradients of any input involved in out
out.backward()

# Gradients are recorded in the variables
print(w0.grad)
print(w1.grad)
```

Outline

Summary

Automatic differentiation procedure

- Uses decomposition of functions in *simple blocks*
- *Forward pass:* Computes function & successive derivatives
- *Backward pass: Back-propagates* derivative of the objective

Summary

Automatic differentiation procedure

- Uses decomposition of functions in *simple blocks*
- *Forward pass:* Computes function & successive derivatives
- *Backward pass: Back-propagates* derivative of the objective

Automatic differentiation toolbox

- Highly efficient and versatile libraries available

Summary

Automatic differentiation procedure

- Uses decomposition of functions in *simple blocks*
- *Forward pass:* Computes function & successive derivatives
- *Backward pass: Back-propagates* derivative of the objective

Automatic differentiation toolbox

- Highly efficient and versatile libraries available

Advanced Differentiation

- Use auto. diff. toolbox to compute *hessians, jacobians,* ...
- Can differentiate through any model, even combinatorial

Extensions

Not covered

- Forward mode of automatic-differentiation, automatic differentiation on complex graph of computations
  $\rightarrow$ see e.g. [Griewank and Walther, 2008]
- Back-propagating sub-gradients for e.g. ReLU activation
  $\rightarrow$ Is auto. diff. well defined ? see [Kakade and Lee, 2018]
- Second-order methods and optimization tricks
  $\rightarrow$ Read [LeCun et al, 1998]

Outline

Computing Second Order Derivative

Until here 'only' gradient computations
Yet derivative of *any* computation available

Computing Second Order Derivative

Until here 'only' gradient computations
Yet derivative of *any* computation available

Question: How to get second order derivative ?

Computing Second Order Derivative

Until here 'only' gradient computations
Yet derivative of *any* computation available

Question: How to get second order derivative ?

Answer: Back-propagate through gradient computations

$$\frac{\partial^2 f}{\partial^2 w} = \frac{\partial}{\partial w}\frac{\partial f}{\partial w}$$

Computing Second Order Derivative

```
from torch import rand, tanh
from torch.autograd import grad

x=rand(1); w0=rand(1); w0.requires_grad = True;

out = tanh(x*w0)

# Flag to record also computations of the gradient
out.backward(create_graph=True)

# Back-propagate through the computations of the gradient
hess = grad(outputs=w0.grad, inputs=w0)
print(hess)
```

Computing Gradients of Multivariate Functions

Functions considered

- Real-valued functions $f : \mathbb{R}^d \to \mathbb{R}$
- Arithmetic operations are matrix-vector products with cost d^2

Computing Gradients of Multivariate Functions

Functions considered

- Real-valued functions $f : \mathbb{R}^d \to \mathbb{R}$
- Arithmetic operations are matrix-vector products with cost d^2

Now Multivariate func. $f : \mathbb{R}^d \to \mathbb{R}^p$, derivative $\nabla f(w) \in \mathbb{R}^{d \times p}$

Computing Gradients of Multivariate Functions

Functions considered

- Real-valued functions $f : \mathbb{R}^d \to \mathbb{R}$
- Arithmetic operations are matrix-vector products with cost d^2

Now Multivariate func. $f : \mathbb{R}^d \to \mathbb{R}^p$, derivative $\nabla f(w) \in \mathbb{R}^{d \times p}$

How to compute gradient ?

Computing Gradients of Multivariate Functions

Functions considered

- Real-valued functions $f : \mathbb{R}^d \to \mathbb{R}$
- Arithmetic operations are matrix-vector products with cost d^2

Now Multivariate func. $f : \mathbb{R}^d \to \mathbb{R}^p$, derivative $\nabla f(w) \in \mathbb{R}^{d \times p}$

How to compute gradient ?

1. Same forward-backward as before
   $\to$ Arithmetic operations are matrix-matrix products, cost pd^2

Computing Gradients of Multivariate Functions

Functions considered

- Real-valued functions $f : \mathbb{R}^d \to \mathbb{R}$
- Arithmetic operations are matrix-vector products with cost d^2

Now Multivariate func. $f : \mathbb{R}^d \to \mathbb{R}^p$, derivative $\nabla f(w) \in \mathbb{R}^{d \times p}$

How to compute gradient ?

1. Same forward-backward as before
   $\to$ Arithmetic operations are matrix-matrix products, cost pd^2
2. Do not compute multi-variate gradient if you only need

$$z :\to \nabla f(w) z \qquad \text{for } z \in \mathbb{R}^p$$

Computing Gradients of Multivariate Functions

Functions considered

- Real-valued functions $f : \mathbb{R}^d \to \mathbb{R}$
- Arithmetic operations are matrix-vector products with cost d^2

Now Multivariate func. $f : \mathbb{R}^d \to \mathbb{R}^p$, derivative $\nabla f(w) \in \mathbb{R}^{d \times p}$

How to compute gradient ?

1. Same forward-backward as before
   $\to$ Arithmetic operations are matrix-matrix products, cost pd^2
2. Do not compute multi-variate gradient if you only need

$$z :\to \nabla f(w) z \qquad \text{for } z \in \mathbb{R}^p$$

$\to$ Amounts to compute gradient of $w :\to z^\top f(w)$

Computing Gradients of Multivariate Functions

Functions considered

- Real-valued functions $f : \mathbb{R}^d \to \mathbb{R}$
- Arithmetic operations are matrix-vector products with cost d^2

Now Multivariate func. $f : \mathbb{R}^d \to \mathbb{R}^p$, derivative $\nabla f(w) \in \mathbb{R}^{d \times p}$

How to compute gradient ?

1. Same forward-backward as before
   $\rightarrow$ Arithmetic operations are matrix-matrix products, cost pd^2
2. Do not compute multi-variate gradient if you only need

$$z :\rightarrow \nabla f(w) z \qquad \text{for } z \in \mathbb{R}^p$$

   $\rightarrow$ Amounts to compute gradient of $w :\rightarrow z^\top f(w)$
   $\rightarrow$ Can solve $\nabla f(w) z = v$ only by calls to $\nabla f(w) z$

Computing Gradients of Multivariate Functions

```
from torch import rand, tanh

X=rand(10, 4); w0=rand(4); w0.requires_grad=True

out = tanh(X.mv(w0))

z = rand(10)

# Add z into the backward operation to convert
# the output to z^T f(w)
out.backward(create_graph=True, gradient=z)

print(w0.grad)
```

Back-propagating through Optimization Procedure

Combinatorial objectives

- Machine learning can involve combinatorial problems
  ex: object segmentation, sentence alignment

Back-propagating through Optimization Procedure

Combinatorial objectives

- Machine learning can involve combinatorial problems
  ex: object segmentation, sentence alignment
- What if features are learned during the optimization ?

Back-propagating through Optimization Procedure

Combinatorial objectives

- Machine learning can involve combinatorial problems
  ex: object segmentation, sentence alignment
- What if features are learned during the optimization ?
- Back-propagates through the optimization iterations

Back-propagating through Optimization Procedure

Structured Prediction **see e.g. [Pillutla et al, 2018]**

- Complex outputs y such as sequences or images

Back-propagating through Optimization Procedure

Structured Prediction **see e.g. [Pillutla et al, 2018]**

- Complex outputs y such as sequences or images
- Prediction of input x given by inference

$$\hat{y} = \max_{\tilde{y}} \phi(x, \tilde{y}; w)$$

where $\phi(x, y; w)$ is the score of label y for input x
$\rightarrow$ Combinatorial problem solved by dynamic programming

Back-propagating through Optimization Procedure

Structured Prediction **see e.g. [Pillutla et al, 2018]**

- Complex outputs y such as sequences or images
- Prediction of input x given by inference

$$\hat{y} = \max_{\tilde{y}} \phi(x, \tilde{y}; w)$$

  where $\phi(x, y; w)$ is the score of label y for input x
  $\rightarrow$ Combinatorial problem solved by dynamic programming

- Given loss on inputs $\ell(\hat{y}, y)$, surrogate objective reads

$$f(w) = \max_{\tilde{y}} \{\phi(x, \tilde{y}; w) + \ell(\tilde{y}, y)\} - \phi(x, y; w)$$

Back-propagating through Optimization Procedure

Structured Prediction see e.g. [Pillutla et al, 2018]

- Complex outputs y such as sequences or images
- Prediction of input x given by inference

$$\hat{y} = \max_{\tilde{y}} \phi(x, \tilde{y}; w)$$

  where $\phi(x, y; w)$ is the score of label y for input x
  $\rightarrow$ Combinatorial problem solved by dynamic programming
- Given loss on inputs $\ell(\hat{y}, y)$, surrogate objective reads

$$f(w) = \max_{\tilde{y}} \{\phi(x, \tilde{y}; w) + \ell(\tilde{y}, y)\} - \phi(x, y; w)$$

- (Sub)-gradient given by back-propagating through the max, i.e., through the dynamic programming procedure