# Machine Learning - Exercise 4

## Companion Slides (adapted from Lucas Beyer)

Paul Voigtlaender Sabarinath Mahadevan

December 6, 2018

# What's the plan?

- Exercise overview
- Deep learning in a nutshell
- Backprop in detail

# Overview

- Goal: implement a simple DL framework from scratch
- Tasks:
  - Compute derivatives (Jacobians)
  - Write code

# Deep Learning in a Nutshell

Given:

- Training data $X = \{x_i\}_{i=1\ldots N}$ with $x_i \in \mathbb{I}$, usually as $X \in \mathbb{R}^{N \times N_I}$
- Training labels $T = \{t_i\}_{i=1\ldots N}$ with $t_i \in \mathbb{O}$

Choose

- Parameterized, (sub-)differentiable function $F(X, \theta) : \mathbb{I} \times \mathbb{P} \to \mathbb{O}$, with
  - typically, input-space $\mathbb{I} = \mathbb{R}^{N_I}$ (generic data), $\mathbb{I} = \mathbb{R}^{3 \times H \times W}$ (images), ...
  - typically, output-space $\mathbb{O} = \mathbb{R}^{N_O}$ (regression), $\mathbb{O} = [0, 1]^{N_O}$ (probabilistic classification), ...
  - typically, parameter-space $\mathbb{P} = \mathbb{R}^{N_P}$
- (Sub-)differentiable criterion/loss $\mathscr{L}(T, F(X, \theta)) : \mathbb{O} \times \mathbb{O} \to \mathbb{R}$

RGB*Height*Width

Find:

$$\theta^* = \underset{\theta \in \mathbb{P}}{\operatorname{argmin}} \mathscr{L}(T, F(X, \theta))$$

Assumption:

$$\mathscr{L}(T, F(X, \theta)) = \frac{1}{N} \sum_{i=1}^{N} \ell(t_i, F(x_i, \theta))$$

# Deep Learning in a Nutshell

Given:

- Training data $X = \{x_i\}_{i=1\ldots N}$ with $x_i \in \mathbb{I}$, usually as $X \in \mathbb{R}^{N \times N_I}$
- Training labels $T = \{t_i\}_{i=1\ldots N}$ with $t_i \in \mathbb{O}$

Choose

- Parameterized, (sub-)differentiable function $F(X, \theta) : \mathbb{I} \times \mathbb{P} \to \mathbb{O}$, with
  - typically, input-space $\mathbb{I} = \mathbb{R}^{N_I}$ (generic data), $\mathbb{I} = \mathbb{R}^{3 \times H \times W}$ (images), ...
  - typically, output-space $\mathbb{O} = \mathbb{R}^{N_O}$ (regression), $\mathbb{O} = [0,1]^{N_O}$ (probabilistic classification), ...
  - typically, parameter-space $\mathbb{P} = \mathbb{R}^{N_P}$
- (Sub-)differentiable criterion/loss $\mathscr{L}(T, F(X, \theta)) : \mathbb{O} \times \mathbb{O} \to \mathbb{R}$

RGB*Height*Width

Find:

$$\theta^* = \underset{\theta \in \mathbb{P}}{\operatorname{argmin}} \mathscr{L}(T, F(X, \theta))$$

Assumption:

$$\mathscr{L}(T, F(X, \theta)) = \frac{1}{N} \sum_{i=1}^{N} \ell(t_i, F(x_i, \theta))$$

Visual Computing Institute

RWTH AACHEN UNIVERSITY

# Deep Learning in a Nutshell

Given:

- Training data $X = \{x_i\}_{i=1\ldots N}$ with $x_i \in \mathbb{I}$, usually as $X \in \mathbb{R}^{N \times N_I}$
- Training labels $T = \{t_i\}_{i=1\ldots N}$ with $t_i \in \mathbb{O}$

Choose

- Parameterized, (sub-)differentiable function $F(X, \theta) : \mathbb{I} \times \mathbb{P} \to \mathbb{O}$, with
  - typically, input-space $\mathbb{I} = \mathbb{R}^{N_I}$ (generic data), $\mathbb{I} = \mathbb{R}^{3 \times H \times W}$ (images), ...
  - typically, output-space $\mathbb{O} = \mathbb{R}^{N_O}$ (regression), $\mathbb{O} = [0, 1]^{N_O}$ (probabilistic classification), ...
  - typically, parameter-space $\mathbb{P} = \mathbb{R}^{N_P}$
- (Sub-)differentiable criterion/loss $\mathscr{L}(T, F(X, \theta)) : \mathbb{O} \times \mathbb{O} \to \mathbb{R}$

RGB*Height*Width

Find:

$$\theta^* = \underset{\theta \in \mathbb{P}}{\operatorname{argmin}} \mathscr{L}(T, F(X, \theta))$$

Assumption:

$$\mathscr{L}(T, F(X, \theta)) = \frac{1}{N} \sum_{i=1}^{N} \ell(t_i, F(x_i, \theta))$$

# Deep Learning in a Nutshell

Given:

- Training data $X = \{x_i\}_{i=1...N}$ with $x_i \in \mathbb{I}$, usually as $X \in \mathbb{R}^{N \times N_I}$
- Training labels $T = \{t_i\}_{i=1...N}$ with $t_i \in \mathbb{O}$

Choose

- Parameterized, (sub-)differentiable function $F(X, \theta) : \mathbb{I} \times \mathbb{P} \to \mathbb{O}$, with
  - typically, input-space $\mathbb{I} = \mathbb{R}^{N_I}$ (generic data), $\mathbb{I} = \mathbb{R}^{3 \times H \times W}$ (images), ...
  - typically, output-space $\mathbb{O} = \mathbb{R}^{N_O}$ (regression), $\mathbb{O} = [0,1]^{N_O}$ (probabilistic classification), ...
  - typically, parameter-space $\mathbb{P} = \mathbb{R}^{N_P}$
- (Sub-)differentiable criterion/loss $\mathscr{L}(T, F(X, \theta)) : \mathbb{O} \times \mathbb{O} \to \mathbb{R}$

RGB*Height*Width

Find:

$$\theta^* = \underset{\theta \in \mathbb{P}}{\operatorname{argmin}} \mathscr{L}(T, F(X, \theta))$$

Assumption:

$$\mathscr{L}(T, F(X, \theta)) = \frac{1}{N} \sum_{i=1}^{N} \ell(t_i, F(x_i, \theta))$$

Visual Computing Institute | RWTH AACHEN UNIVERSITY

# Backprop

Take derivative with respect to $\theta$

$$D_\theta \frac{1}{N}\sum_{i=1}^{N}\ell(t_i, F(x_i,\theta)) = \frac{1}{N} D_\theta \sum_{i=1}^{N}\ell(t_i, F(x_i,\theta))$$

$$= \frac{1}{N}\sum_{i=1}^{N} D_\theta F(x_i,\theta)\cdot D_F \ell(t_i, F(x_i,\theta))$$

Chain rule

Assumption:

$F$ is hierarchical: $F(x_i,\theta) = f_1(f_2(f_3(\ldots x_i \ldots, \theta_3), \theta_2), \theta_1)$

$D_{\theta_1} F(x_i,\theta) = D_{\theta_1} f_1(f_2,\theta_1)$

$D_{\theta_2} F(x_i,\theta) = D_{\theta_2} f_2(f_3,\theta_2)\cdot D_{f_2} f_1(f_2,\theta_1)$

$D_{\theta_3} F(x_i,\theta) = D_{\theta_3} f_3(\ldots,\theta_3)\cdot D_{f_3} f_2(f_3,\theta_2)\cdot D_{f_2} f_1(f_2,\theta_1)$

Where $f_2 = f_2(f_3(\ldots x_i \ldots, \theta_3), \theta_2)$ etc.

# Backprop

Take derivative with respect to $\theta$

$$D_\theta \frac{1}{N}\sum_{i=1}^{N} \ell(t_i, F(x_i,\theta)) = \frac{1}{N} D_\theta \sum_{i=1}^{N} \ell(t_i, F(x_i,\theta))$$

$$= \frac{1}{N}\sum_{i=1}^{N} D_\theta F(x_i,\theta)\cdot D_F \ell(t_i, F(x_i,\theta))$$

Chain rule

Assumption:
$F$ is hierarchical: $F(x_i,\theta) = f_1(f_2(f_3(\dots x_i \dots, \theta_3), \theta_2), \theta_1)$
$D_{\theta_1} F(x_i,\theta) = D_{\theta_1} f_1(f_2, \theta_1)$
$D_{\theta_2} F(x_i,\theta) = D_{\theta_2} f_2(f_3, \theta_2)\cdot D_{f_2} f_1(f_2, \theta_1)$
$D_{\theta_3} F(x_i,\theta) = D_{\theta_3} f_3(\dots, \theta_3)\cdot D_{f_3} f_2(f_3, \theta_2)\cdot D_{f_2} f_1(f_2, \theta_1)$
Where $f_2 = f_2(f_3(\dots x_i \dots, \theta_3), \theta_2)$ etc.

Visual Computing Institute | RWTH AACHEN UNIVERSITY

# Jacobians

The loss:

$$D_F \ell(t_i, F(x_i, \theta)) = \begin{pmatrix} \partial_{F_1} \ell \\ \vdots \\ \partial_{F_{N_F}} \ell \end{pmatrix} \in \mathbb{R}^{N_F \times 1}$$

The functions (modules):

$$f(z, \theta) = \begin{pmatrix} f_1(z_1, \dots, z_{N_z}; \theta) & \dots & f_{N_f}(z_1, \dots, z_{N_z}; \theta) \end{pmatrix} \in \mathbb{R}^{1 \times N_f}$$

$$D_z f(z, \theta) = \begin{pmatrix} \partial_{z_1} f_1 & \cdots & \partial_{z_1} f_{N_f} \\ \vdots & \ddots & \vdots \\ \partial_{z_{N_z}} f_1 & \cdots & \partial_{z_{N_z}} f_{N_f} \end{pmatrix} \in \mathbb{R}^{N_z \times N_f}$$

Visual Computing Institute | RWTH AACHEN UNIVERSITY

# Jacobians

The loss:

$$D_F \ell(t_i, F(x_i, \theta)) = \begin{pmatrix} \partial_{F_1} \ell \\ \vdots \\ \partial_{F_{N_F}} \ell \end{pmatrix} \in \mathbb{R}^{N_F \times 1}$$

The functions (modules):

$$f(z, \theta) = \begin{pmatrix} f_1(z_1, \dots, z_{N_z}; \theta) & \dots & f_{N_f}(z_1, \dots, z_{N_z}; \theta) \end{pmatrix} \in \mathbb{R}^{1 \times N_f}$$

$$D_z f(z, \theta) = \begin{pmatrix} \partial_{z_1} f_1 & \cdots & \partial_{z_1} f_{N_f} \\ \vdots & \ddots & \vdots \\ \partial_{z_{N_z}} f_1 & \cdots & \partial_{z_{N_z}} f_{N_f} \end{pmatrix} \in \mathbb{R}^{N_z \times N_f}$$

Visual Computing Institute | RWTH AACHEN UNIVERSITY

# Modules

Looking at module $f_2$:

$$D_{\theta_3} F(x_i,\theta) = [D_{\theta_3} f_3(\ldots,\theta_3)] \underbrace{\underbrace{[D_{f_3} f_2(\overbrace{\overset{\text{input}}{\overbrace{f_3}}, \theta_2)]}^{\text{output}}}_{\text{Jacobian wrt. input}} \underbrace{[D_{f_2} f_1(f_2,\theta_1)]}_{\text{grad\_output}}}_{\text{grad\_input}}$$

Three (core) functions per module:
fprop: compute the output $f_i(z,\theta_i)$ given the input $z$ and current parameters $\theta_i$
grad_input: compute $D_z f_i(z,\theta_i) \cdot \mathrm{grad\_output}$
grad_param: compute $\nabla_{\theta_i} = D_{\theta_i} f_i(z,\theta_i) \cdot \mathrm{grad\_output}^{\mathrm{T}}$
Typically:
fprop caches its input and/or output for later reuse
grad_input and grad_param are combined into single bprop function to share computation

# Usage/Training

$net = [f1, f2, \ldots, f_{N_f}], \ell = criterion$
**for** $Xb, Tb$ in batched $X, T$ **do**
    $z = Xb$
    **for** module in net **do**
        $z = module.fprop(z)$
    **end for**
    $costs = \ell.fprop(z, Tb)$
    $\partial z = \ell.bprop([\frac{1}{N_B} \ldots \frac{1}{N_B}])$
    **for** module in reversed(net) **do**
        $\partial z = module.bprop(\partial z)$
    **end for**
    **for** module in net **do**
        $\theta, \partial\theta = module.params(), module.grads()$
        $\theta = \theta - \lambda \cdot \partial\theta$
    **end for**
**end for**

# Example: Linear aka. Fully-connected module

$$f(z, W, b) = z \cdot W + b \in \mathbb{R}^{1 \times N_f}$$

Where $z \in \mathbb{R}^{1 \times N_z}$, $W \in \mathbb{R}^{N_z \times N_f}$, $b \in \mathbb{R}^{1 \times N_f}$, and
$\text{grad\_output} = D_f \ell(f(z, W, b))^T \in \mathbb{R}^{N_f \times 1}$
The gradients are

- $\mathbb{R}^{N_z \times N_f} \ni \text{grad\_W} = z^T \cdot \text{grad\_output}^\text{T}$
- $\mathbb{R}^{N_f \times 1} \ni \text{grad\_b} = \text{grad\_output}$
- $\mathbb{R}^{N_z \times 1} \ni \text{grad\_input} = W \cdot \text{grad\_output}$

# Gradient Checking

Crucial debugging method!
Compare Jacobian computed by finite differences using the $\mathrm{fprop}$ function to Jacobian computed by the $\mathrm{bprop}$ function.
Advice: Use (small) random input $x$, and $h_i = \sqrt{eps}\max(x_i, 1)$.
Finite-difference: first column of Jacobian as:

$$x_- = \begin{pmatrix} x_1 - h_1 & x_2 & x_3 & \dots & x_{N_x} \end{pmatrix}$$

$$x_+ = \begin{pmatrix} x_1 + h_1 & x_2 & x_3 & \dots & x_{N_x} \end{pmatrix}$$

$$J_{\bullet,1} = \frac{\mathrm{fprop}(x_+) - \mathrm{fprop}(x_-)}{2h_1}$$

Backprop: first row of Jacobian as:

$$\mathrm{fprop}(x)$$

$$J_{1,\bullet} = \mathrm{bprop}\begin{pmatrix} 1 & 0 & 0 & \dots & 0 \end{pmatrix}$$

# Mini-Batching

Linear layer (without mini-batching)

$$f(z, W, b) = z \cdot W + b$$

$z \in \mathbb{R}^{1\times N_z}, W \in \mathbb{R}^{N_z \times N_f},\ b \in \mathbb{R}^{1\times N_f}$
Stack $z$ into mini-batch matrix with batch size $N \Rightarrow z \in \mathbb{R}^{N\times N_z}$
Now the multiplication $z \cdot W$ can be performed for all examples in one pass and $b$ can be added by broadcasting (repeating) $b$ to $\hat{b}$

$$f(z, W, b) = \begin{pmatrix} \text{---} & f_1 & \text{---} \\ \vdots & \vdots & \vdots \\ \text{---} & f_N & \text{---} \end{pmatrix} = \begin{pmatrix} z_{1,1} & \cdots & z_{1,N_z} \\ \vdots & \ddots & \vdots \\ z_{N,1} & \cdots & z_{N,N_z} \end{pmatrix} \cdot \begin{pmatrix} W_{1,1} & \cdots & W_{1,N_f} \\ \vdots & \ddots & \vdots \\ W_{N_z,1} & \cdots & W_{N_z,N_f} \end{pmatrix} + \begin{pmatrix} \text{---} & b & \text{---} \\ \vdots & \vdots & \vdots \\ \text{---} & b & \text{---} \end{pmatrix}$$

# Rule-of-thumb result on MNIST

$\text{Linear}(28 \times 28, 10),\ \text{Softmax}$ should give $\pm 750$ errors.

$\text{Linear}(28 \times 28, 200),\ \text{tanh},\ \text{Linear}(200, 10),\ \text{SoftMax}$ should give $\pm 250$ errors.

Typical learning rates $\lambda \in [0.1, 0.01]$

Typical batch-sizes $N_B \in [100, 1000]$

Initialize weights as $\mathbb{R}^{M \times N} \ni W \sim \mathcal{N}(0, \sigma = \sqrt{\frac{2}{M+N}})$ and $b = 0$

Don't forget data pre-processing, here at least divide values by 255 (max pixel value).