

Advanced Machine Learning - Exercise 3

Deep learning essentials



RWTHAACHEN
UNIVERSITY

Introduction

What's the plan?

Exercise overview

Deep learning in a nutshell

Backprop in (painful) detail



Introduction

Exercise overview

Goal: implement a simple DL framework

Tasks:

- Compute derivatives (Jacobians)
- Write code

You'll need some help. . .



Introduction

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 $t_i \in \mathbb{O}$.



Introduction

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:

- Parametrized, (sub-)differentiable function $F(X, \theta) : \mathbb{I} \times \mathbb{P} \mapsto \mathbb{O}$, 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 $\mathbb{P} = \mathbb{R}^{N_P}$.



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:

- Parametrized, (sub-)differentiable function $F(X, \theta) : \mathbb{I} \times \mathbb{P} \mapsto \mathbb{O}$, 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 $\mathcal{L}(T, F(X, \theta)) : \mathbb{O} \times \mathbb{O} \mapsto \mathbb{R}$

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:

- Parametrized, (sub-)differentiable function $F(X, \theta) : \mathbb{I} \times \mathbb{P} \mapsto \mathbb{O}$, 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 $\mathcal{L}(T, F(X, \theta)) : \mathbb{O} \times \mathbb{O} \mapsto \mathbb{R}$

Find:

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

Introduction

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:

- Parametrized, (sub-)differentiable function $F(X, \theta) : \mathbb{I} \times \mathbb{P} \mapsto \mathbb{O}$, 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 $\mathcal{L}(T, F(X, \theta)) : \mathbb{O} \times \mathbb{O} \mapsto \mathbb{R}$

Find:

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

Assumption:

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

Backprop

$$\begin{aligned} D_{\theta} \frac{1}{N} \sum_{i=1}^N \ell(t_i, F(x_i, \theta)) &= \frac{1}{N} \sum_{i=1}^N D_{\theta} \ell(t_i, F(x_i, \theta)) \\ &= \frac{1}{N} \sum_{i=1}^N D_F \ell(t_i, F(x_i, \theta)) \circ D_{\theta} F(x_i, \theta) \end{aligned}$$

Backprop

$$\begin{aligned} D_{\theta} \frac{1}{N} \sum_{i=1}^N \ell(t_i, F(x_i, \theta)) &= \frac{1}{N} \sum_{i=1}^N D_{\theta} \ell(t_i, F(x_i, \theta)) \\ &= \frac{1}{N} \sum_{i=1}^N D_F \ell(t_i, F(x_i, \theta)) \circ D_{\theta} F(x_i, \theta) \end{aligned}$$

Assumption:

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

Backprop

$$\begin{aligned} D_{\theta} \frac{1}{N} \sum_{i=1}^N \ell(t_i, F(x_i, \theta)) &= \frac{1}{N} \sum_{i=1}^N D_{\theta} \ell(t_i, F(x_i, \theta)) \\ &= \frac{1}{N} \sum_{i=1}^N D_F \ell(t_i, F(x_i, \theta)) \circ D_{\theta} F(x_i, \theta) \end{aligned}$$

Assumption:

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

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

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

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

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

Backprop

Jacobians

The Loss:

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



Backprop

Jacobians

The Loss:

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

The functions (modules):

$$f(z, \theta) = \begin{pmatrix} f_1((z_1 \dots z_{N_z}), \theta) \\ \vdots \\ f_{N_f}((z_1 \dots z_{N_z}), \theta) \end{pmatrix}$$

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

Backprop

Modules

Looking at module f_2 :

$$D_{\theta_3} F(x_i, \theta) = \underbrace{\underbrace{[D_{f_2} f_1(f_2, \theta_1)]}_{\text{grad_output}} \underbrace{[D_{f_3} f_2(f_3, \theta_2)]}_{\text{Jacobian wrt. input}}}_{\text{grad_input}} [D_{\theta_3} f_3(\dots, \theta_3)]$$

The diagram illustrates the backpropagation through module f_2 . The equation shows the gradient of the total loss F with respect to the parameters θ_3 . The term $[D_{f_3} f_2(f_3, \theta_2)]$ is labeled as the Jacobian with respect to the input of f_2 . This Jacobian is multiplied by the gradient of f_1 with respect to its output, $[D_{f_2} f_1(f_2, \theta_1)]$, which is labeled as grad_output. The product of these two terms is labeled as grad_input, representing the gradient flowing into module f_2 . The final term, $[D_{\theta_3} f_3(\dots, \theta_3)]$, represents the gradient of the final module f_3 with respect to its parameters θ_3 .



Backprop

Modules

Looking at module f_2 :

$$D_{\theta_3} F(x_i, \theta) = \underbrace{\underbrace{[D_{f_2} f_1(f_2, \theta_1)]}_{\text{grad_output}} \underbrace{[D_{f_3} f_2(f_3, \theta_2)]}_{\text{Jacobian wrt. input}}}_{\text{grad_input}} [D_{\theta_3} f_3(\dots, \theta_3)]$$

The diagram shows the chain rule for the derivative of the loss function F with respect to the parameters θ_3 . The expression is $D_{\theta_3} F(x_i, \theta) = [D_{f_2} f_1(f_2, \theta_1)] [D_{f_3} f_2(f_3, \theta_2)] [D_{\theta_3} f_3(\dots, \theta_3)]$. Brackets indicate the flow of gradients: a bracket labeled 'grad_output' spans the first two terms, a bracket labeled 'Jacobian wrt. input' spans the second term, and a bracket labeled 'grad_input' spans the entire product. Additionally, a bracket labeled 'input' is above the second term, and a bracket labeled 'output' is above the first term.

Three (core) functions per module:

fprop: compute the output $f_i(z, \theta_i)$ given the input z and current parametrization θ_i .

grad_input: compute $\text{grad_output} \cdot D_z f_i(z, \theta_i)$.

grad_param: compute $\nabla_{\theta_i} = \text{grad_output} \cdot D_{\theta_i} f_i(z, \theta_i)$.

Backprop

Modules

Looking at module f_2 :

$$D_{\theta_3} F(x_i, \theta) = \underbrace{\left[D_{f_2} f_1(f_2, \theta_1) \right]}_{\text{grad_output}} \underbrace{\left[D_{f_3} f_2(\overbrace{f_3}^{\text{input}}, \theta_2) \right]}_{\text{Jacobian wrt. input}} \left[D_{\theta_3} f_3(\dots, \theta_3) \right]$$

grad_input

Three (core) functions per module:

`fprop`: compute the output $f_i(z, \theta_i)$ given the input z and current parametrization θ_i .

`grad_input`: compute $\text{grad_output} \cdot D_z f_i(z, \theta_i)$.

`grad_param`: compute $\nabla_{\theta_i} = \text{grad_output} \cdot D_{\theta_i} f_i(z, \theta_i)$.

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.



Backprop

(Mini-)Batching

Remember: $\frac{1}{N} \sum_{i=1}^N D_F \ell(t_i, F(x_i, \theta)) \circ D_\theta F(\dots)$ where $D_F \ell = (\partial_{F_1} \ell \ \dots \ \partial_{F_{N_F}} \ell) \in \mathbb{R}^{1 \times N_F}$

Reformulate as matrix-vector operations allows computation in a single pass:

$$\begin{pmatrix} \frac{1}{N} & \dots & \frac{1}{N} \end{pmatrix} \cdot \begin{pmatrix} \partial_{F_1} \ell(t_1, F(x_1, \theta)) & \dots & \partial_{F_{N_F}} \ell(t_1, F(x_1, \theta)) \\ \vdots & & \vdots \\ \partial_{F_1} \ell(t_N, F(x_N, \theta)) & \dots & \partial_{F_{N_F}} \ell(t_N, F(x_N, \theta)) \end{pmatrix} \in \mathbb{R}^{N \times N_F}$$

Backprop

Usage/training

```
net = [f1, f2, ...], l = criterion  
for Xb, Tb in batched X, T:
```

```
    z = Xb
```

```
    for module in net:
```

```
        z = module.fprop(z)
```

```
    costs = l.fprop(z, Tb)
```

```
 $\partial z = l.bprop([\frac{1}{N_B} \dots \frac{1}{N_B}])$ 
```

```
    for module in reversed(net):
```

```
         $\partial z = module.bprop(\partial z)$ 
```

```
    for module in net:
```

```
         $\theta, \partial \theta = module.params(), module.grads()$ 
```

```
         $\theta = \theta - \lambda \cdot \partial \theta$ 
```

Backprop

Example: Linear **aka.** Fully-connected **module**

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

Where $z \in \mathbb{R}^{1 \times N_z}$, $W \in \mathbb{R}^{N_z \times N_f}$, and $b \in \mathbb{R}^{1 \times N_f}$.

The gradients are:

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



Backprop

Gradient checking

Crucial debugging method!

Compare Jacobian computed by finite differences using `fprop` function to Jacobian computed by `bprop` function.

Advice: Use (small) random input x , and $h_i = \sqrt{\text{eps}} \max(x_i, 1)$.

Finite-difference: first column of Jacobian as:

$$\begin{aligned}x_- &= (x_1 - h_1 \quad x_2 \quad \dots \quad x_{N_x}) \\x_+ &= (x_1 + h_1 \quad x_2 \quad \dots \quad x_{N_x}) \\J_{\bullet,1} &= \frac{\text{fprop}(x_+) - \text{fprop}(x_-)}{2h_1}\end{aligned}$$

Backprop: first row of Jacobian as:

$$\begin{aligned}J_{1,\bullet} &= \text{bprop}(1 \quad 0 \quad \dots \quad 0)\end{aligned}$$

Backprop

Rule-of-thumb results on MNIST

Linear(28*28, 10), SoftMax should give ± 750 errors.

Linear(28*28, 200), Tanh, Linear(200, 10), SoftMax should give ± 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.)

Merry Christmas and a happy New Year!

Also, good luck for the exercise =)

