Advanced Machine Learning - Exercise 3

Deep learning essentials

RWTH AACHEN UNIVERSITY

What's the plan?

- Exercise overview
- Deep learning in a nutshell
- Backprop in (painful) detail

Exercise overview

Goal: implement a simple DL framework

Tasks:

- Compute derivatives (Jacobians)
- Write code

You'll need some help...

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 $t_i \in \mathbb{O}$.

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:

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

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:

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

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

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

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

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

Assumption:

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

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

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

Jacobians

The Loss:

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

Jacobians

The Loss:

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

The functions (modules):

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

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

Modules

Looking at module f_2:

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

Modules

Looking at module f_2:

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

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 `grad_output` $\cdot\, D_z f_i(z, \theta_i)$.

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

Backprop

Modules

Looking at module f_2:

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

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

(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 $D_F\ell = \begin{pmatrix}\partial_{F_1}\ell & \dots & \partial_{F_{N_F}}\ell\end{pmatrix} \in \mathbb{R}^{1\times N_F}$

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

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

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)

  ∂z = l.bprop([1/N_B ... 1/N_B])
  for module in reversed(net):
    ∂z = module.bprop(∂z)

  for module in net:
    θ, ∂θ = module.params(), module.grads()
    θ = θ − λ · ∂θ
```

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:

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

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 $h_i = \sqrt{eps}\max(x_i, 1)$.

Finite-difference: first column of Jacobian as:

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

Backprop: first row of Jacobian as:

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

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

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 =)