Homework 2

Balint Negyesi

24-02-2022

[Extended!] Due date: 17th of March, 2022, before the start of the lecture
Submission via email: b.negyesi@tudelft.nl.
Submissions should consist of a *short* `.pdf` report of your findings, and an offline, compilable copy of the jupyter notebook.

In this homework we are concerned with regression problems and their optimization. Throughout the exercises you are allowed to use the linear algebra and random libraries of `numpy` and the neural network classes (together with their respective optimizations) provided in `handout_week_1.ipnyb`. In case of the latter, be careful to use an appropriate loss function instead of cross entropy. For instance, under the PyTorch API you can find a pre-implemented mean-squared error loss module in

```
from torch.nn import MSELoss
```

Furthermore, you can change the optimizer's hyperparameters (learning rate$\leftarrow \eta$, momentum$\leftarrow \alpha$) by passing the following code snippets to `self.optimizer()`

```
torch.optim.SGD(self.parameters(), lr=eta, momentum=alpha)
```

Solve the following exercises.

Ex. 1 (Automatic differentiation – `numpy` only)

Consider the following deep neural network

$$x \ni \mathbb{R} \mapsto \mathrm{Id} \circ z^{(L+1)}(\cdot|\theta^{(L+1)})\varphi^{(L)} \circ z^{(L)}(\cdot|\theta^{(L)}) \circ \cdots \circ \varphi^{(1)} \circ z^{(1)}(x|\theta^{(1)}) \in \mathbb{R},$$

where each hidden layer has unit width $p_l = 1$, $l = 1, \dots L$. Consider hyperbolic tangent activations $\varphi^{(l)}(z) = \tanh(z)$, $\forall l = 1, \dots, L$. Fix some $L > 1$, and random **non-constant zero** $w^{(l)}, b^{(l)} \in \mathbb{R}$, $\forall l = 1, \dots, L+1$ and compute the network's derivative $\frac{\mathrm{d}\Phi(x|\Theta)}{\mathrm{d}x}$ at $x_0 = 1$:

(a) analytically, using the chain rule, in terms of activations and their affine combinations;
(b) by finite differences with $h \in \{10^{-2}, 10^{-4}, 10^{-6}, 10^{-8}\}$;
(c) by automatic differentiation.

Compare the results of the latter two.

Ex. 2 (Optimization methods – `numpy` only)

We seek to approximate a scalar-valued function $f : [-1, 1] \to \mathbb{R}$ by means of a linear model fitted on order p polynomials. Thus, the set of basis functions (features) in the linear model is $\{1, x, x^2, x^3, \dots, x^p\}$ and our *approximations* are given by

$$\Psi(x|\beta \coloneqq (\beta_0, \dots, \beta_p)) \coloneqq \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_p x^p. \tag{1}$$

The hand of Adam Smith reaches out to us and tells us that the true mapping of f may be defined by

$$f(x) = \left(e^{\cos\left(0.2(2\pi x)^2\right)} - 1\right)^2$$

However, our sampling device is crooked and can only draw data points with an $\epsilon \sim \mathcal{N}(\mu = 0, \sigma^2 = 0.01^2)$ error term. Therefore the data points accessible are distributed as follows

$$y_m = \left(e^{\cos\left(0.2(2\pi x_m)^2\right)} - 1\right)^2 + \epsilon_m. \tag{2}$$

Our goal is to find the *best* possible approximation of the form (1) which minimizes the *mean-squared error* of M measurements drawn in the fashion of (2)

$$\beta^* = \underset{\beta \in \mathbb{R}^{p+1}}{\arg\min} \frac{1}{M} \sum_{m=1}^{M} (y_m - \Psi(x_m|\beta))^2. \tag{3}$$

Fix $M = 10^4, p = 20$. For the iterative methods, use random normal initializations.

(a) Find the true minimizer of the model using the derivation in Lecture 1.

(b) Solve the minimization problem (3) by implementing the iterative Newton's method. *Hint: max_ iter* $\leq 10^2$ *suffices.*

(c) Solve the minimization problem (3) by implementing raw Gradient Descent with no stochasticity involved. *Hint: choose max_ iter* $\leq 10^5$ *and* $\eta^k = 10^{-1}$.

(d) Solve the minimization problem (3) by implementing Stochastic Gradient Descent with mini batch size `batch_size`. *Hint: choose max_ iter* $\leq 10^3$, $\eta^k = 10^{-1}$ *and* $B \in \{1, 64, 256, 1024\}$ – *no need to report on all.*

(e) Evaluate and compare your results obtained with Ex. 2a–Ex. 2b–Ex. 2c–Ex. 2d in terms of

- accuracy compared to the *truth* given by (2); and compared to the *true minimizer* given by (3);
- runtime;
- in case of the iterative schemes, draw convergence plots on both errors above;
- change the initialization β^0 in Ex. 2c–Ex. 2d to $\widehat{\beta}^{\text{Newton}}$, how do your solutions change?

(f) Based on your findings try to motivate why SGD is more common in machine learning practices than Newton's method.

Ex. 3 (Neural network regression – `PyTorch` modules given in `handout_week_1.ipnyb`)

Instead of the linear model in (1), consider neural network parametrizations of the form

$$\Phi(x|\Theta) := \mathrm{Id} \circ z^{(L+1)}(\cdot|\theta^{(L+1)})\varphi^{(L)} \circ z^{(L)}(\cdot|\theta^{(L)}) \circ \cdots \circ \varphi^{(1)} \circ z^{(1)}(x|\theta^{(1)}).$$

Fix $p_l = 100, \forall l = 1, \ldots, L$. Use $\varphi^{(l)} = \mathrm{relu}(x), \forall l = 1, \ldots, L$ activations in each hidden layer. Solve the same minimization problem (3) with these neural network parametrizations in the following cases. *Hint: choose* $M_{train} = 10^4$, $B = 2^{10}$, $\eta^k = 10^{-2}$.

(a) Using a shallow neural network ($L = 1$).

(b) Using a deep neural network of $L = 3$ hidden layers.

(c) Compare your results with Ex. 2a.

(d) What happens if you include *momentum* in the SGD iterations with $\alpha = 0.9$?