Okvir za automatsko optimiranje funkcija sastavljenih od diferencijabilnih građevnih elemenata

10. listopada 2024.

1 Uvod

1.1 Derivacija kompozicije funkcija

Neka je varijabla y definirana preko varijable x i složene funkcije $F \coloneqq f_n \circ f_{n-1} \circ \cdots \circ f_1$ ovako:

$$\begin{aligned} y &\coloneqq F(x) && (1)\\ &= (f_n \circ f_{n-1} \circ \cdots \circ f_1)(x) && (2)\\ &= f_n(f_{n-1}(\dots f_1(x)). && (3) \end{aligned}$$

Za $n = 2$ je $y = f_2(f_1(x))$ i, prema pravilu deriviranja složene funkcije (engl. *chain rule*), vrijedi

$$F'(x) = f_2'(f_1(x))f_1'(x). \tag{4}$$

Ako uvedemo varijablu $h \coloneqq f_1(x)$, onda je $y = f_2(h)$ i jednadžbu (4) možemo izraziti ovakvim oznakama:

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial h}\frac{\partial h}{\partial x}. \tag{5}$$

Pravilo deriviranja složene funkcije može se poopćiti i na kompoziciju većeg broja

funkcija. Za općeniti n, računanje varijable y može se razložiti na niz koraka:

$$h_1 = f_1(x), \tag{6}$$
$$h_2 = f_2(h_1), \tag{7}$$
$$\vdots \tag{8}$$
$$y = h_n = f_n(h_{n-1}), \tag{9}$$

gdje su h_k, $k = 1..n$, međurezultati.

Neka $F_k \coloneqq f_k \circ f_{k-1} \circ \cdots \circ f_1$. $F = F_n$ možemo rekurzivno izraziti ovako:

$$F_n = f_n \circ F_{n-1} \tag{10}$$

Uzastopnom primjenom pravila deriviranje složene funkcije iz izraza (4) dobivamo

$$\begin{aligned} F_n'(x) &= f_n'(F_{n-1}(x))F_{n-1}'(x) \\ &= f_n'(F_{n-1}(x))f_{n-1}'(F_{n-2}(x))F_{n-2}'(x) \\ &\vdots \\ &= f_n'(F_{n-1}(x))f_{n-1}'(F_{n-2}(x)) \cdots f_2'(f_1(x))f_1'(x) \\ &= f_n'(h_{n-1})f_{n-1}'(h_{n-2}) \cdots f_2'(h_1)f_1'(x). \end{aligned}$$

Ako koristimo oznake

$$\frac{\partial h_k}{\partial h_{k-1}} = f_k'(h_{k-1}), \tag{11}$$

to možemo i ovako izraziti:

$$F'(x) = f_n'(h_{n-1})f_{n-1}'(h_{n-2}) \cdots f_2'(h_1)f_1'(x) \tag{12}$$
$$= \frac{\partial y}{\partial h_n}\frac{\partial h_n}{\partial h_{n-1}} \cdots \frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial x}. \tag{13}$$

Funkcije koje preslikavaju vektore u vektore. Može se pokazati da isti izrazi vrijede i za funkcije kojima su ulazi i izlazi vektori, samo što su onda derivacije jakobijani, a množenje je matrično množenje. Ako $\boldsymbol{y} = f(\boldsymbol{x})$, onda je jakobijan $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}$ matrica s ovako definiranim elementima:

$$\left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)_{[i,j]} \coloneqq \frac{\partial \boldsymbol{y}_{[i]}}{\partial \boldsymbol{x}_{[j]}}. \tag{14}$$

Ako $f\colon \boldsymbol{x} \mapsto y$ preslikava vektor u skalar, jakobijan $\frac{\partial y}{\partial \boldsymbol{x}}$ je vektor redak, a gradijent $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) =$ je vektor stupac iste dimenzije kao $\boldsymbol{x}$.

1.2 Gradijentni spust

Metoda gradijentnog spusta najjednostavnija je metoda optimizacije prvog reda. Označimo funkciju koju optimiramo sa f. Tada prema Taylorovom razvoju prvog reda vrijedi:

$$f(\boldsymbol{x} + \Delta\boldsymbol{x}) = f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathsf{T}} \Delta\boldsymbol{x}. \tag{15}$$

Ako sada za pomak $\Delta\boldsymbol{x}$ uvrstimo negativan gradijent pomnožen sa hiperparametrom ϵ dobivamo:

$$\Delta\boldsymbol{x} = -\epsilon \cdot \nabla f(\boldsymbol{x}), \tag{16}$$

$$f(\boldsymbol{x} + \Delta\boldsymbol{x}) = f(\boldsymbol{x}) - \epsilon \cdot \nabla f(\boldsymbol{x})^{\mathsf{T}} \nabla f(\boldsymbol{x}). \tag{17}$$

Vrijednost funkcije f bi očigledno trebala opadati, ipak u slučaju prevelike vrijednosti hiperparametra ϵ pomak će također biti velik i funkcija bi mogla porasti. Hiperparametar ϵ u kontekstu optimizacije zovemo korakom optimizacije, a u kontekstu strojnog učenja korakom ili stopom učenja. Pojednostavljeno ideja gradijentnog spusta je sljedeća: ako gradijent pokazuje smjer najvećeg porasta funkcije, s ciljem minimizacije idemo se kretati malim koracima u smjeru negativnog gradijenta.

Algoritam 1: Gradijentni spust

Ulaz: Stopa učenja ϵ
Ulaz: Početna točka $\boldsymbol{x}$
$k \leftarrow 1$
dok *nije ispunjen uvjet zaustavljanja* **radi**
  izračunaj gradijent: $\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$
  napravi ažuriranje: $\boldsymbol{x} \leftarrow \boldsymbol{x} - \epsilon \boldsymbol{g}$
  $k \leftarrow k + 1$
kraj

1.3 Automatska diferencijacija

Biblioteke s podrškom za automatsku diferencijaciju kao što su TensorFlow ili PyTorch neki matematički izračun predstavljaju preko računskog grafa. To je usmjereni graf u kojem čvorovi predstavljaju matematičke operacije, a bridovima putuju podaci predstavljeni kao tenzori varijabilnih dimenzija. Korisnik željeni izračun zadaje putem programskog sučelja (python) koristeći i kombinirajući osnovne gradivne blokove. Jedan gradivni blok obično predstavlja jednu matematičku operaciju. Svaki gradivni blok mora znati za zadani ulaz izračunati izlaz, ali i izračunati gradijent izlaza svoje funkcije prema bilo kojem od mogućih ulaza.

Ispis 1: Sučelje gradivnog bloka

```
class Node:
```

```
    def forward(self, input):
        # računa izlaz na temelju ulaza
        pass
    def backward(self, output_grad=1):
        # računa gradijent po ulazu uz dani gradijent po izlazu
        pass
```

Ispis 2: Implementacija gradivnog bloka za prirodni logaritam

```
class Log(Node):
    def forward(self, input):
        assert input > 0
        self.input = input
        return np.log(self.input)

    def backward(self, output_grad=1):
        return output_grad * (1 / self.input)
```

1.4 Parcijalne derivacije vektorskih funkcija i logistička regresija

http://www.zemris.fer.hr/~ssegvic/du/lab0.shtml

2 Zadaci

Implementirati okvir za automatsko optimiranje funkcija sastavljenih od lanca diferencijabilnih građevnih elemenata. Implementirati građevne elemente potrebne za optimizaciju potpuno povezane neuronske mreže. Provesti postupak treniranja modela neuronske mreže na podatkovnom skupu MNIST i prikazati rezultate.

2.1 Faza 1: optimizacija funkcije skalarne varijable

Zadaci:

1. Implementirati građevne elemente prema sučelju Node koji računaju sljedeće operacije: $x \mapsto x^c$, $x \mapsto \exp(x)$, $x \mapsto \ln(x)$, $x \mapsto x + c$, $x \mapsto c \cdot x$, $x \mapsto c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n$.
2. Implementirati unaprijedni i unatražni prolaz za zadani lanac građevnih elemenata. Implementirati postupak gradijentnog spusta. Napisati glavni program koji traži minimume sljedećih funkcija: $f(x) = \ln\big(\ln\big(\exp\big((x-1)^2\big)+2\big)\big)$, $f(x) = (x^2-1)^3$, $f(x) = \ln\left(1+\exp(x^2)\right)$

Rok: 25.11.2024.

2.2 Faza 2: optimizacija parametrizirane funkcije vektorske varijable

Zadaci:

1. Prilagoditi postojeće implementacije računskih čvorova (građevnih blokova) na način da podržavaju vektorski ulaz i izlaz (pripaziti na unatražni prolaz - jakobijan). Neka potenciranje i množenje vektora bude po elementima.
2. Omogućiti učenje funkcije pomoću računskih čvorova koji mogu sadržavati parametre koji se mogu optimirati. Metodu `backward` proširiti tako da uz gradijent po ulazu vraća i listu parova "(parametri, gradijent po parametrima)". Ako čvor sadrži druge čvorove, treba vratiti gradijent po svom ulazu i konkatenaciju listi parova "(parametri, gradijent po parametrima)" čvorova koje sadrži. Napomena: Kako bi se mogli ažurirati parametri iz liste, oni, i ako su skalari, moraju biti tipa `numpy.ndarray`.
3. Ostvariti funcionalnost potrebnu za inicijalizaciju parametara.
4. Implementirati računske čvorove:
   - prijenosne funkcije $\sigma(x) = \frac{1}{1+\exp(-x)}$ i $\mathrm{ReLU}(x) = \max(0, x)$,
   - afinu transformaciju (potpuno povezani sloj) $\boldsymbol{x} \mapsto \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$.
   - $\mathrm{softmax}(\boldsymbol{s}) = \frac{\exp(\boldsymbol{s})}{\sum_i \exp(\boldsymbol{s})_{[i]}}$, koji ulaz $\boldsymbol{s}$, kategoričke logite, preslikava u vektor koji predstavlja kategoričku razdiobu.
5. Implementirati računski čvor `SoftmaxCrossEntropyWithLogits`. Metoda `forward` na ulazu prima logite i ciljnu oznaku, a računa gubitak unakrsne entropije.
6. Implementirati metodu zasnovanu na postupku gradijentnog spusta koja će za zadani skup podataka, funkciju gubitka i model optimirati parametre modela. Metoda bi trebala podržavati "običnu" i stohastičku varijantu gradijentnog spusta. Prilikom implementacije gradijentnog spusta možete napraviti unaprijedni i unatražni prolaz za svaki podatak posebno, a nakon toga napraviti ažuriranje sa uprosječenim gradijentom.
7. Naučiti model logističke regresije za podatke iz nulte vježbe kolegija duboko učenje http://www.zemris.fer.hr/~ssegvic/du/src/data.py.
8. Usporediti brzinu i točnost gradijentnog spusta i stohastičkog gradijentnog spusta.
9. Vizualizirati graf gubitka na skupu za učenje u ovisnosti o broju iteracija/epoha.
10. Vizualizirati napredak decizijske krivulje.

Rok: 10.1.2024.

Literatura

[1] Randal J. Barnes. Matrix differentiation. 2010. URL https://atmos.washington.edu/~dennis/MatrixCalculus.pdf.

[2] Christopher Olah. Calculus on computational graphs: Backpropagation. 2015. URL http://colah.github.io/posts/2015-08-Backprop/.

[3] Christopher Olah. Visual information theory. 2015. URL http://colah.github.io/posts/2015-09-Visual-Information/.

[4] Siniša Šegvić. Podsjetnik na logističku regresiju, gradijentni spust, Python i NumPy. 2016. URL http://www.zemris.fer.hr/~ssegvic/du/lab0.shtml.

[5] Jan Šnajder i Bojana Dalbelo Bašić. *Strojno učenje*. 2014.