Zadání druhého projektu SUI 2024/25

Karel Beneš

30. října 2024

Changelog: 30. října 2024 Zadání zveřejněno

Cílem druhého projektu v SUI je vytvořit implementovat minimalistickou knihovnu pro tvorbu neuronových sítí. Prostředí pro vývoj a evaluaci algoritmů je k dispozici na Githubu[1]. Termín odevzdání je neděle 17. listopadu 2024 ve 23:59:59, odevzdává se ve Studisu.

V rámci projektu máte vytvořit několik dílčích náležitostí:

1. Sčítání (agregaci) prvků tensoru – tato operace je v projektu jedinečná, protože je jejím výstupem skalár.
2. Jednoduché operace sčítání, odečítání a násobení tensorů po prvcích.
3. Jedinou nelinearitu – ReLU.
4. Maticové násobení.
5. Zpětné šíření chyby pro všechny tyto operace.

Rozhraní a náležitosti řešení Ústředním objektem v celém tomto projektu je tensor, instancee třídy `Tensor`. Tento zapouzdřuje samotné hodnoty v tensoru (ve formě `numpy` pole) a navíc s sebou nese informaci o tom, jak tímto tensorem šířit chybu, a potažmo, jaký je gradient chyby podle tohoto tensoru. Hodnota tensoru a odpovídající gradient stejných rozměrů (`value` a `grad`) jsou považovány za jeho rozhraní a nelze měnit jejich název ani význam. Informace o zpětném šíření chyby (`back_op`) je plně k dispozici pro Vaše kreativní řešení, můžete ji používat jakkoliv, pouze neměňte její název, tak bylo zachováno formální rozhraní.

Ve třídě `Tensor` je potřeba doimplementovat metodu pro zpětné šíření chyby (`Tensor.backward()`): Pokud je zavolána bez parametru `deltas`, zkontroluje, že je volána nad tensorem o jediném prvku, tento chápe jako chybu, a začne ji zpětně šířit. Při volání s parametrem `deltas` chápe tento jako chyby vzhledem k jednotlivým prvkům tensoru, uloží ho jako gradint vzhledem k tomuto tensoru a opět ho zpětně šíří. Zpětné šíření chyby lze řešit pomocí rekurze.

Ostatní operace jsou implementovány jako jednotlivé funkce. Očekává se od nich, že vytvoří nový tensor, naplní ho odpovídající hodnotou a dodají mu vhodnou `back_op`, za tím účelem je možno vytvářet vlastní objekty či třídy dle libosti.

Při implementaci můžete předpokládat, že:

- všechny tensory vstupující do operací budou formálně právě dvoudimenzionální a
- při budování neuronových sítí budou data považována za sloupcové vektory a při dopředném průchodu sítí tedy budou násobeny do matic zprava.

Pokud Vaše řešení zvládně i obecnější situace, ničemu to nevadí.

Implementovat budete v jazyce Python, odevzdávat budete jediný soubor `sui_torch.py`, ten musí dodat všechnu implementaci ve Vašem projektu. Při tvorbě Vašeho řešení nesmíte použít žádnou knihovnu pro automatickou diferenciaci nebo implementaci neuronových sítí. Naopak, je povinnost použít `numpy` pro implementaci jednotlivých operací, ve Vašem řešení se nesmí objevit žádné cyklení přes prvky tensorů.

Všechna místa pro implementaci jsou vyznačena pomocí `raise NotImplentedError()`, navíc můžete podle potřeby přidávat vlastní funkce či třídy. Do již přítomného kódu nezasahujte.

[1] https://github.com/ibenes/sui-torch, prosím klonovat, tak aby Vaše implementace zůstala soukromá.

Kontrola správnosti řešení Pro základní ověření správnosti je k dispozici základní sada jednotkových testů (`test.py`). Pokud k vývoji přistoupíte na základě studia (tj. nikoliv metodou pokus – omyl), lze předpokládat, že Vám tato sada může stačit. Dále je k dispozici skript, který pomocí Vašeho kódu natrénuje několik malých neuronových sítí (`training.py`). Jeho zdárný průchod bude součástí vyhodnocení projektů, je tedy nezbytné, aby s ním Vaše řešení bylo kompatibilní.

Doporučení pro řešení

- Pro zpětné šíření chyby potřebujete přinejmenším vědět, z čeho se nový tensor vypočítal.
- Maticové násobení je nejsložitější operace v projektu. Zvažte, jestli od jeho implementace začít, nebo s ní počkat, až budete mít prošlápnutou backpropagaci s ostatními.
- Trénovací skript dává smysl spouštět až po implementaci všech částí projektu.