Neura - programátorská dokumentace

1. Úvod

Neura je python knihovna, která implementuje základní stavební kameny neuronových sítí. Její vhodné využití je například aproximace matematických funkcí nebo binární klasifikace jednoduchých problémů.

Knihovna používá knihovnu numpy pro všechny operace s maticemi a jedná se o jedinou použitou knihovnu. Numpy neimplementuje GPU akceleraci, takže kód ve výsledku celý běží na CPU. V důsledku toho knihovna není vhodná pro velké projekty, protože je trénink časově náročný.

Syntaktický standard PEP8 je dodržován v celé knihovně.

2. Rozdělení

Soubory neury jsou strukturovány jako python knihovna. Obsahuje složku examples, ale veškeré její funkcionality jsou v pěti souborech:

- activation functions,
- layers,
- loss_functions,
- optimizers,
- network.

Každý soubor, kromě network.py, v sobě mimo jiné obsahuje abstraktní třídu pro snadnou implementaci nových funkcionalit. Network již další rozšiřování nepodporuje a nabízí tak pouze sekvenční sítě.

3. Aktivační funkce

Všechny aktivační funkce se nachází v souboru *activation_functions.py*, kde se nachází i abstraktní třída *AbstractActivationFunction*. Tato třída deklaruje dvě funkce: *forward* a *derived*. Funkce forward popisuje chování funkce při výpočtu výstupu, zatímco funkce *derived* počítá gradient při zpětné propagaci chyby. Neura v základu implementuje dvě aktivační funkce: *Sigmoid* a *LeakyReLU*.

3.1. Sigmoid

Sigmoid je aktivační funkce, která pro funkci forward používá rovnici

$$\sigma_{(x)} = \frac{1}{1 + e^{-x}},$$

z čehož tedy vyplívá, že obor hodnot je

$$H_\sigma = (0,1).$$

Funkce je tedy vhodná především ke klasifikačním úlohám a z tohoto důvodu byla i vybrána.

Konstruktor této třídy nepřijímá žádné argumenty, ale k dalšímu použití v knihovně je vždy třeba vytvořit instanci.

3.2. LeakyReLU

LeakyReLU je aktivační funkcí z rodiny *Rectified Linear Unit* aktivačních funkcí. Byla vybrána proto, že při správném zvolení parametrů je možné s ní nahradit i základní *ReLU*, ale nabízí větší všestrannost.

Při dopředné propagaci se řídí podle vzorce

$$\sigma_{(x)} = \begin{cases} x & if\ x > 0 \\ ax & otherwise \end{cases}$$

a její obor hodnot tvoří tedy všechna reálná čísla. Proto ji lze dobře využít k aproximaci matematických funkcí.

Tato třída přijímá v konstruktoru argument *gradient_of_negative_input*, který je ve vzorci pojmenován *a*. Tento parametr určuje, jakým způsobem se bude nakládat s negativními vstupy.

4. Loss funkce

Loss funkce se nachází společně s jejich abstraktní třídou v souboru loss_functions.py. Abstraktní třída *AbstractLossFunction* deklaruje také pouze metodu *forward* a *derived*. Funkce *forward* vypočítá dopřednou propagaci a vypočítá její průměr, zatímco funkce *derived* vypočítá gradient loss funkce. Pro oba výpočty jsou používány parametry: výstupní hodnota sítě a předpokládaná výstupní hodnota sítě (label). Knihovna implementuje dvě loss funkce: MSE a Binary Cross-Entropy.

4.1. MSE

MSE, nebo také mean squared error loss function, je jedna z nejzákladnějších loss funkcí vůbec. Pro dopřednou propagaci používá vzorec

$$\sigma_{(x,y)} = \frac{1}{n}\sum_{k=1}^{n}(y_k - x_k)^2\ ,$$

kde y_k je k-tá předpokládaná hodnota (label) a x_k je k-tá hodnota, predikovaná sítí.

4.2. Binary cross-entropy

Binary cross-entropy je typ chybové funkce, užitečný k predikci v klasifikačních problémech. Řídí se vzorcem

$$\sigma_{(x,y)} = -\frac{1}{n}\sum_{k=1}^{n} y_k \log(x_k) + (1 - y_k)\log(1 - x_k)\ ,$$

kde y_k je k-tá předpokládaná hodnota (label) a x_k je k-tá hodnota, predikovaná sítí.

5. Vrstvy

Kód pro vrstvy se nachází v soboru layer.py společně s abstraktní třídou *AbstractLayer*, která implementuje dvě základní metody *forward* a *backpropagate*. Metoda *forward* popisuje, jakým způsobem má vrstva nechat skrz sebe projít data. Funkce *backpropagate* vytvoří gradienty proměnných, které později aplikuje optimizer.

Neura implementuje pouze jeden typ vrstvy a to DenseLayer. Pro implementaci nových vrstev je možné snadno použít abstraktní třídu.

5.1. DenseLayer

Dense layer je nejzákladnější vrstvou neuronových sítí. Jako argumenty do konstruktoru přijímá velikost vstupu, velikost výstupu a instanci aktivační funkce. Na základě těchto dat se vygenerují váhy a prahy pomocí Xavier inicializace. Dopředná propagace u neuronů vypadá takto

$$y = \sigma\left(b + \sum_{k=1}^{n} x_k \cdot w_k\right),$$

kde b je práh (bias), w_k je k-tá váha, x_k je k-tý input, σ je aktivační funkce a n je počet vstupů do neuronu. Pro zrychlení celého procesu jsou počítány operace jako násobení a sčítání matic. Poté se aktivační funkce aplikuje pouze na jednotlivé prvky matice.

Ve funkci *backpropagate* se počítají gradienty vah a prahů, které později optimizér aplikuje.

6. Optimizéry

Optimizéry v knihovně neura slouží k aplikaci gradientů, vypočítaných ve zpětné propagaci chyby. Abstraktní třída *AbstractOptimizer* implementuje pouze funkci *update_weights*, která přijímá list vrstev jako argument. Knihovna disponuje dvěma optimizery: SGD a MomentumSGD.

6.1. SGD

SGD neboli stochastic gradient descent, je nejzákladnějším algoritmem ve strojovém učení s učitelem. V konstruktoru definujeme pouze proměnnou *learning_rate*, která definuje, jak rychle se model bude učit. Poté změnu jednotlivých proměnných popisuje vzorec

$$p_{t+1} = p_t - \frac{\partial L}{\partial p_t}\alpha,$$

kde p_t je nynější hodnota některého parametru, α je learning rate, L je loss funkce a p_{t+1} je nová hodnota parametru. V případě knihovny neura bude výraz $\frac{\partial L}{\partial p_t}$ již vypočten a optimizér ho bude tedy pouze aplikovat.

6.1. MomentumSGD

Momentum SGD je pouze zjednodušený název stochastic gradient descent s momentem. Jediný rozdíl mezi SGD a MomentumSGD je, že momentum příjmá do svého konstruktoru ještě jeden parametr: momentum.

Momentum nechává poslední změnu ovlivnit nadcházející. Hodnota momenta určuje, jak moc bude momentum ovlivňovat další změnu, a proto se převážně volí v rozmezí (0,1). Pokud zvolíme hodnotu momenta 0, dostaneme vlastně SGD. Momentum se řídí rovnicemi

$$p_{t+1} = p_t - V_t$$
$$V_t = V_{t-1}\beta + \frac{\partial L}{\partial p_t}\alpha,$$

kde p_t je nynější hodnota některého parametru, α je learning rate, L je loss funkce a p_{t+1} je nová hodnota parametru, V_t je změnový vektor a β je hodnota momenta. Třída v sobě ukládá vždy nejnovější V_t aby ho mohla později použít.

7. Network

Network je posledním komponentem knihovny. Všechny funkcionality jednotlivých prvků dává do jedné velké třídy, která zprostředkovává proces učení, testování i predikce dat.

Všechny funkce třídy network jsou podrobně rozepsány v uživatelské dokumentaci

Zdroje

Inspirace jmen a struktury knihovny: https://www.tensorflow.org/api_docs/python/tf/keras
Inspirace pro psaní některých funkcí, včetně backpropagation: https://nnfs.io/
Ověření rovnic a dalších faktických informací: https://en.wikipedia.org/