LSTM.py

Ősosztály tartalmazza az LSTM hálózatot, melyet felparaméterezett konstruktorával meglehet hívni. Az osztály tartalmaz egy konstruktort, melyből a hálózat paramétereit lehet megadni és egy Forward függény, amelyet prediktálásra, és betanításra lehet majd használni.

Ma az LSTM hálózatom a Pytorch nn.Module osztályt származtatja le. Pytoch esetén ez az osztály képezi bármely neurális hálózat alapmodellét, ezáltal feltétel, hogy az én modellem és ennek egy leszármazottja legyen.

Az osztály leszármaztatást követően megadhatom a hálózatom rétegeit. A felépítésem a következő: bemeneti réteg: Itt 2 dimenziós bemeneti inputokat továbbítok a hálózat felé. /\*milyen inputokat használok ilyesmi\*/

1 réteg LSTM réteg. 16 neuronnal. Feltélezésem az volt, hogy minden absztrakcióhoz felveszek egy neuront. Az LSTM rétegek számát 1nek határoztam meg, a gyorsabb predikciók érdekében. /\*LSTM felépítéses ábra, RNN ábra\*/

Fully Connected Layer az érdekében, hogy az outputomat formázni tudjam és megteremtsem az elvárt dimenziónalitást bevezettem egy teljesen összekötött réteget, amely az output formázására szolgál.

A forward metódussal tudok inputokat továbbítani a hálózatnak. LSTM esetén meg kell a cella aktuális állapotát is, a hidden layeren felűl. Ezek iniciális állapota mindig nulla lesz.

Pred.py

A fő osztálya a hálózatomnak. Itt példányosítom a korábbiakban bemutatott LSTM hálózatot és itt paraméterezem fel. A Pred osztály feladata, hogy managelje az inputokat, a hálózatnak irányítsa, tanítsa és bármikor tudjon predikciókat adni.

Az LSTM hálózat az osztály konstruktorában kerül példányosításra és itt indul a preporcessing folyamat is. A dataset feldolgozása is itt kezdődik. A Feldolgozáshoz egy előre elkészített Dataset.py adathalmazt fogok használn. Ahhoz hogy kisebb és könnyebben kezelhető számokkal dolgozzak és tanítsak a hálózatnak, Egy MinMaxScaler-t fogok hasznáni. A skálázó segítségével normalizálhatom a datasettemet.

A normalizált datasettet ezután újraformázom numpy segítségével. Olyan formátumra akarok változtatni, amelyet könnyen és folyamatosan tudok továbbítani a hálózatnak.

A tanítás során a ‘csúszóablak’ módszert alkalmazom, miszerint a prediktálni kívánt távolság értéke szerint ‘eltolom’ annyi elemmmel a tömb elemeit. És az eltolt elemeket végét használom target ként. /\*szebben megfogalmazni.\*/

Lehetőségem van már előre elkészített modelleket használni vagy újakat készíteni .PTH formátumban. A jövőben, mikor inferenciára akarom használni, képen lévő objektumok helyzetének prediktálását, akkor szükséges lesz egy előre elkészített modell betöltésére.

Amennyiben új modellt akarok generálni, szükség lesz annak betanítására, amelyet a train metódus fog végezni.

Minden hálózat a 0-ról indul. Itt kell megadni a szükséges paramétereket, miszerint mennyivel dolgozzon előre, milyen gyakorisággal végezzen predikciót, milyen hosszú adathalamzon. Ez esetben további preprocessingre van szükség. 2 adathalamzt határozok meg, az inp - bemeneti értékeket és a tar - cél értékeket. Ezen értékeket a hálózat számára feldolgozható formává alakítom, azaz tensorokká.

Miután a tensorokat meghatároztam, a meg kell adnom az optimizert és a loss- veszteség függvényemet. Mivel egy regressziós problémával dolgozok ideális választás az MSEloss loss függvény. Optimizernek Adamet válaszottam, overall jó optimalizáló.

/\*Tanítás folyamatát még lehet részletezni\*/

A megadott epoch számokon tensorokat szolgáltatok a hálózatnak, azok feldolgozása alapján határozza meg a súlyait és módosítja azokat az optimezer és a loss alapján. /\*backpropagate\*/

A tanítás végén a kész modellt elmentem az aktuális dátum szerint, hogy később inferenciára használhassuk.

A pred osztály következő metódusa a ‘predict’. A predict metódus a kapott inputra végez predikciót. A metódus hívása során a modelt evaluációs állapotba állítom és kikapcsoljuk a gradienst. Természetesen a betanítás folyamata ilyekor már rég véget ért, viszont