Protein szekvenciák elemzése Long Short Term Memory segítségével

Fischer Kornél

2021 május

1 A korábbi félév feladatának lezárása

A félév elején folytattuk az eredeti adathalmazzal a korábbi munkát, és tovább javítottuk az XGBoostot gridsearchet alkalmazva. Sok próbálkozás után látszott, hogy még a legjobb esetben is csak megközelíteni sikerült a Random Forest legjobb eredményét. Emiatt úgy döntöttünk, hogy visszatérünk a Random Foresthez, és megvizsgáljuk, hogy a modellek pontosságának mi a várható értéke, valamint mekkora a szórása, és ugyanezt megcsináltuk az XGBoostra is. Miután mindkét modellen megtaláltuk a legjobb hiperparaméterezést, csináltunk 10-10 darab véletlen train-test vágást. Minden vágásnál lefuttattuk a modelleket, és megnéztük a predikciók négyzetes eltérését.

Azt volt megfigyelhető, hogy hasonló a várható értékük és a szórásuk is, ebből pedig arra következtettünk, hogy a két modell lényegében ugyanolyan jól oldja meg a feladatot. Tovább kísérletezve látszott, hogy a Random Forest már kisebb tanító halmazon is eléri ezt a jó pontosságot. Ezen tények egy lehetséges magyarázata, hogy a feladat nem volt elég komplex ahhoz, hogy az XGBoost erőssége érvényesüljön a Random Forest-tel szemben. Ezt felismerve új adatbázis után néztünk, és az alapján egy új, nehezebb feladatot fogalmaztunk meg.

2 Az új adatbázis

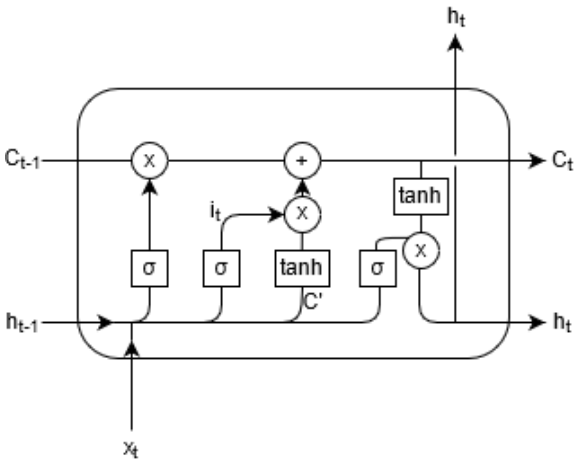
Az új adatbázisunkban szintén protein szekvenciák szerepeltek, továbbá egy 7 lehetésges értéket felvevő címke, ami azt jelölte, hogy a sejten belül hol helyezkedik el az adott protein [1]. Az adathalmazban eredetileg 700.000 rekord volt, viszont nem tudtuk mindet felhasználni a tanítás során, meghaladta volna a számítási kapacitásainkat. Az első teszteket egy kisebb, 7000 rekordos adathalmazon futtattuk, és később egyre növeltük az adathalmaz méretét, egészen 300.000 rekordig. Erre a feladatra a Long Short Term Memory nevű rekurrens neurális hálót alkalmaztuk.

3 Rekurrens Hálók

A neurális hálókat elképzelhetjük úgy, mint számítási gráfokat. Ezekben a gráfokban nem engedünk meg köröket, így tudunk backpropagationt számolni. A rekurrens hálóknál az az ötletünk, hogy megengedünk a gráfban hurkokat. Ezeken a hurkokon tudunk információt küldeni a hálóban lépések között. A hurkok miatt a gráf már nem lesz DAG, de mégis működhet a backpropagation, ha elképzeljük, hogy azt a hálórészletet, ahol a hurok van, többször egymás mellé lerajzolva. Így ismét DAG-ot kaptunk, és használhatjuk a láncszabályt a backpropagation számítására.

Az alkalmazott rekurrens háló az úgynevezett Long Short Term Memory, röviden LSTM [2,3], ennek egyik hasznos tulajdonsága, hogy jól kezeli a hosszútávú függőségeket, emiatt előszerettel használják szekvenciák vizsgálatakor.

Az LSTM legfontosabb része a cell state. Ez úgy képzelhető el, mint egy futószalag, ami végigmegy az egész láncon, és minden lépésben csak kétszer interaktál a háló többi részével. Az LSTM képes törölni vagy hozzáadni információt a cell statehez, ezt olyan neuronok segítségével teszi, amiknek az aktivációs függvénye vagy szigmoid, vagy hiperbolikus tangens.



1. ábra: Az LSTM modell

Egy szigmoid aktivációs függvényt használó neuron outputja 0 és 1 között lesz, ez a szám azt fogja eldönteni, hogy az információ mekkora része menjen a cell state-be. Nem érdemes mindent eltárolnunk, szelektálnunk kell. Ha túl sok információt tárolnánk, akkor hosszadalmas lenne a gradiens számolása. Az LSTM-nek három külön neuronja van, ami kontrollálja a cell statet, ezeket az következőkben ismertetjük.

Az LSTM elején egy szigmoid neuron azt dönti el, mennyi információt dobjunk

el a cell stateből. Ezt az előző hidden state kimenetéből és a mostani bemenetből számítja ki, és ezek alapján a C_{t-1} számaihoz rendel hozzá 0 és 1 közti értékeket.

Ezután azt döntjük el, milyen új információt fogunk tárolni a cell state-ben. Ez két részből áll, az elsőben egy szigmoid réteggel eldöntjük, melyik értékeket fogjuk frissíteni, ezt jelöljük i_t-vel. Utána egy hiperbolikus tangens neuronnal előállítunk olyan jelölt vektorokat, amik a cell statehez adhatók, ezeket jelöljük C'-vel.

Az előző kapuk döntései alapján frissítjük a régi C_{t-1}-et, megalkotva C_t-t. C_{t-1}-ből elfelejtjük, amit úgy döntöttünk, el fogjuk, majd hozzáadjuk $i_t * C'$-t.

Végül pedig eldöntjük, mi lesz a kimenet. A state cell-ből egy szigmoid réteggel kiszedjük azt a rész, amit szeretnénk visszaadni, egy tanh kapuval beállítjuk -1 és 1 közé az értéket, majd a szigmoid kapu outpujával ezt megszorozzuk, és az így kapott h_t vektort továbbadjuk a következő fázisnak.

Az LSTM egy változata a Bidirectional LSTM, ennek esetén van még egy neuron réteg, ami a szekvenciát visszafele dolgozza fel. Így a tanulás során egyszerre van olyan információnk, amit a kapott szekvencia eddigi részéből nyerhettünk ki, és abból a részből is, ami majd következni fog. Sok feladatnál az ilyen típusú LSTM-ek jobban teljesítenek a sima társaiknál.

A háló outputja az utolsó h_t lesz, amit mi a fent ismertetett feladatban arra használtunk, hogy klasszifikációs feladatot oldjunk meg vele.

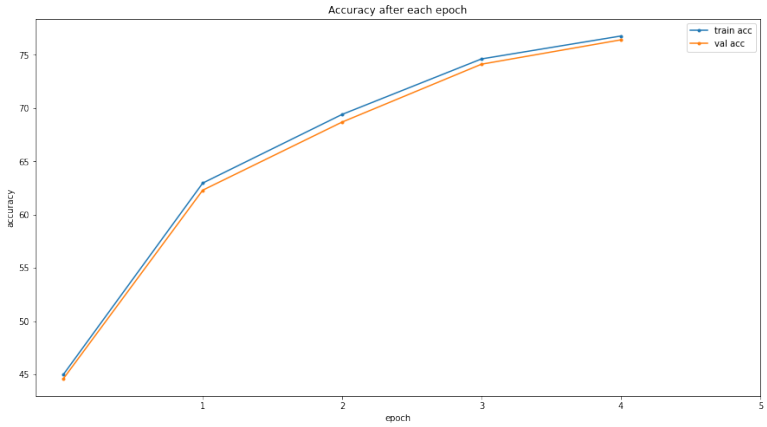
4 A feladat megoldása

Az előfeldolgozást hasonlóan kezdtük mint az előző félévben, tovább javítva az ott már kipróbált és működő technikákat. A szekvenciák közül eldobtuk azokat, amik 1500-nál hosszabbak voltak, hogy könnyítsük az adatok kezelését. Ezen szekvenciák száma 10.000 körül volt, így továbbra is bőven maradt rekordunk. A megmaradt szekvenciákat aminosavakra bontottuk, majd az aminosavakat egy legyártott szótár segítségével kódoltuk természetes számokkal 1 és 24 között, meghagyva a 0-t egy különleges karakternek, amit a padeléskor később használtunk.

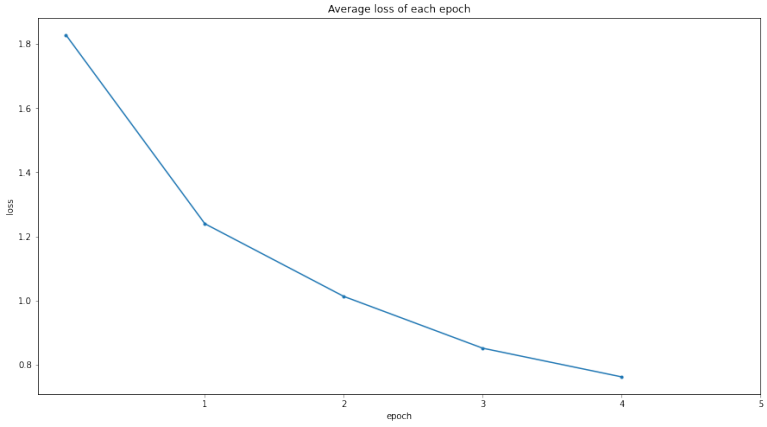
Az LSTM-nek tenzorként kell odaadnunk a szekvenciákat ahhoz, hogy feltudja dolgozni őket. A fehérjék egy pandas dataframe-ben voltak tárolva. Hossz szerinti sorrendbe helyeztük őket, hogy amikor veszünk egy batchméretnyi szekvenciát, azok körülbelül ugyanolyan hosszúak legyenek. A túl rövid szekvenciákat padeltük 0-val, viszont ennek a megközelítésnek köszönhetően kevés padelésre volt szükség. A háló elején egy embedding réteget alkalmaztunk, ami beágyazta az input vektorokat egy olyan térbe, aminek a dimenzióját mi választhattuk meg. Ez a legtöbb tanítás során egy 50 dimenziós tér volt. Így az LSTM bemenetként egy olyan tenzort kapott, aminek a három dimenziója a batchméret, a szekvencia-hossz, és a beágyazás dimenziója. További fontos paraméter a hálóban a hidden dimension, ami az LSTM h_t kimenetének dimenzióját szabályozta.

A háló végén egy fully connected linear réteg volt, ami softmax segítségével választott a 7-féle osztály közül egyet. Loss függvénynek kereszt-entrópiát használtunk.

Amíg a hiperparamétereknek kerestünk megfelelő beállításokat, kisebb adathalmazon dolgoztunk, csak pár ezer szekvencián. Miután találtunk olyan paramétereket, amik jónak tűntek, csináltunk egy tanítást 270.000 rekordon, 30.000 rekordot használva tesztelésre. Az LSTM-nek sikerült elérnie a 0,767 illetve 0,764-es pontosságot a tanító illetve teszt halmazon, 0.76-os veszteség mellett [2. ill. 3. ábra].



2. ábra: A pontosság alakulása



3. ábra: A veszteség alakulása

5 További vizsgálatok

Felmerült, hogy vajon miféle módon oszlanak el a címkék, nem lehet-e, hogy az adatok egy érték körül összpontosulnak. Az adathalmazon az átlag 3.06 volt, a szórás pedig 2.00. Az LSTM sokkal jobban teljesített annál, mintha véletlenül választottunk volna címkét.

Amikor egy szótárral kódoltuk az aminosavakat, definiáltunk egy sorrendet közöttük, lett ami közelebb került egymáshoz, más távolabb. Lehetséges volt, hogy ez megzavarja a beágyazást és az LSTM-et a tanulás során. Emiatt ki kellett próbálnunk, hogy mennyire változnak az eredmények akkor, ha a szótárat másképp adjuk meg. Véletlenül megkevertük az aminosavak sorrendjét a szótárban, újrakódoltuk, és lefuttattunk több tesztet is. Az eredmények nagyjából megegyeztek a korábbi mérésekkel. Ebből arra következtettünk, hogy ez a kialakult sorrend nem okoz gondot a klasszifikálásban.

Az első pár méréseknél az látszott, hogy az első 7-8 epoch során a modell nem nagyon javult, mindig beállt egy 0,24-0,25-ös pontosságra, és nem mozdult ki onnan, majd hirtelen a következő 2-3 epoch alatt sokat javult. Ilyenkor bár a pontosság nem javult, a loss viszont monoton csökkent, vagyis a modell kis mértékben ilyenkor is javított. Ezt a furcsa viselkedést valószínűleg a használt Adam optimizer okozta. Egy lehetséges magyarázat lehet erre a jelenségre, hogy beállt egy lokális optimumra, és sokáig tartott, amíg onnan kilökődött, és tovább tudott javulni. Valószínű ez a jelenség nem lett volna ilyen látványos, ha más optimizert használtunk volna. Ugyanakkor amikor már nagyobb adathalmazzal tanítottuk a modellt, ez a probléma nem jelentkezett többé, sokkal gyorsabban tanult az LSTM, és kevesebb epoch is elég volt a nagy pontosság eléréséhez.

6 Folytatás

A következő félévben további modelleket lehetne kipróbálni hasonló feladatokra, például a BERT modelt, ami jól teljesít természetes nyelvfeldolgozó feladatokon. Klasszifikációs feladat helyett unsupervised tanulást lehetne kipróbálni.

7 Irodalom

[1] Protein Locations: `https://www.kaggle.com/swlew369/protein-locations`

[2] Hochreiter, Schmidhuber. Long Short Term Memory: `https://www.researchgate.net/publication/13853244_Long_Short-term_Memory`

[3] Understanding LSTM Networks: https://colah.github.io/posts/2015-08-Understanding-LSTMs/