



» Лекционен курс

» Въведение в генеративния ИИ



RNN >

Регистрация

Registration for: ВГИИ

Четвърта седмица



Short

URL:<https://tinyurl.com/24j394f2>

Start at: 17-05-2024 08:45

Status: CREATED

End at: 17-05-2024 09:45

Registrations: 0



Увод

- » Досега изследвахме генеративни модели, които включват **латентни променливи** - вариационни автоенкодери (VAE)
- » В тези модели се въвежда **нова случайна променлива**, като така лесно се взема проба и моделът се научава как да декодира тази променлива обратно в оригиналния домейн.



Увод

- » Сега ще насочим вниманието си **към авторегресивните модели** - семейство от модели, които опростяват проблема с генеративното моделиране, като го третираат като **последователен процес**.
- » Авторегресивните модели обуславят прогнозите върху **предишни стойности в последователността**, а не върху латентна случайна променлива.
- » Следователно те се опитват **явно да моделират разпределението**, генериращо данни, а не негово приближение (както в случая с VAE).



Увод

- » В тази лекция ще разгледаме авторегресивния модел:
 - > Мрежа с дълга краткосрочна памет (**LSTM**)
- » Ще приложим:
 - > LSTM към текстови данни
- » В следващата лекция ще разгледаме подробно друг много успешен авторегресивен модел, наречен **трансформатор**.



Една история



The Literary Society for Troublesome Miscreants (LSTM)



Х мразеше работата си като надзирател в затвора. Той прекарваше дните си в наблюдение на затворниците и нямаше време да следва истинската си страст **да пише разкази**. Не му достигаше вдъхновение и трябваше да намери начин да **генерира ново съдържание**.

Един ден му хрумва брилянтна идея, която ще му позволи да създаде нови научно-фантастични произведения (негов стил), като същевременно ще занимава затворниците – ще накара затворниците колективно да пишат историите вместо него.

Нарича новото общество Литературно общество за проблемни злодеи (The Literary Society for Troublesome Miscreants или LSTM) (фигурата).



The Literary Society for Troublesome Miscreants (LSTM)



Затворът се състои само от една голяма килия, съдържаща 256 затворници.

Всеки затворник има мнение за това как трябва да продължи настоящата история.

Всеки X публикува най-новата дума от романа си в килията и задачата на затворниците е индивидуално да актуализират мненията си за текущото състояние на историята въз основа на новата дума и мненията си от предишния ден.

Всеки затворник използва специфичен мисловен процес, за да актуализира собственото си мнение, което включва балансиране на информацията от новата входяща дума и мненията на други затворници с техните собствени предишни вярвания.



The Literary Society for Troublesome Miscreants (LSTM)



Първо, те решават каква част от вчерашното мнение искат да забравят, като вземат предвид новата дума и мненията на други затворници в килията.

Те също така използват тази информация, за да формират нови мисли и да решат до каква степен искат да ги смесят със старите версии, които са избрали да продължат от предишния ден.

Това формира новата версия на затворника за деня.



The Literary Society for Troublesome Miscreants (LSTM)



Затворниците обаче са потайни и не винаги казват на съкилийниците си цялото си мнение. Всеки от тях също използва последната избрана дума и мненията на другите затворници, за да реши каква част от мнението си иска да разкрие.

Когато X иска килията да генерира следващата дума в поредицата, всеки от затворниците казва своите разкриваеми мнения на пазача на вратата, който комбинира тази информация, за да реши в крайна сметка следващата дума да бъде добавена към края на романа.

След това тази нова дума се връща обратно в клетката и процесът продължава, докато бъде завършена цялата история.



The Literary Society for Troublesome Miscreants (LSTM)



За да обучи затворниците и пазача X подава кратки поредици от думи, които е написал преди това, в килията и следи дали избраната от затворниците следваща дума е правилна. Той ги информира за тяхната точност и постепенно те започват да се учат как да пишат истории в неговия собствен уникален стил.

След много повторения на този процес X установява, че системата е доста завършена в генерирането на реалистично изглеждащ текст. Доволен от резултатите, той публикува колекция от генерираните приказки в новата си книга.



Long Short-Term Memory Network (LSTM)



Определение

- » LSTM е определен тип **повтаряща (recurrent) се невронна мрежа (RNN)**.
- » RNN съдържат **повтарящ се слой** (или клетка), който е в състояние да обработва последователни данни, като прави своя собствен изход на определена времева стъпка част от входа на следващата времева стъпка.



Проблем в RNN

- » Когато RNN бяха въведени за първи път, повтарящите се слоеве бяха много прости и се състоеха единствено от **оператора \tanh** , който гарантира, че информацията, предадена между времевите стъпки, е мащабирана между -1 и 1 .
- » Въпреки това беше показано, че този подход страда от проблема с **изчезващия градиент** и не се мащабира добре към дълги последователности от данни.



Въвеждане на LSTM

- » Клетките LSTM бяха въведени за първи път през 1997 г.
- » LSTM **не страдат от същия проблем с изчезващия градиент**, изпитван от стандартните RNN, и могат да бъдат обучени на последователности, които са дълги стотици стъпки.
- » Оттогава архитектурата на LSTM е адаптирана и подобрена.



Приложение

- » LSTM се използват за решаване на широк набор от проблеми, **включващи последователни данни**, като например:
 - > Прогнозиране на времеви редове.
 - > Анализ на настроение.
 - > Аудио класификация.
- » В тази лекция ще използваме LSTM за генериране на текст.



Наборът от данни за рецепти



Изтегляне набора данни

```
bash scripts/download_kaggle_data.sh hugodarwood epirecipes
```

Ще използваме набора от данни Epicurious Recipes, който е достъпен чрез Kaggle.

Това е набор от над **20 000 рецепти**, с придружаващи **метаданни** като хранителна информация и списъци със съставки.

Можете да изтеглим набора от данни, като изпълним скрипта за изтегляне на набор данни от Kaggle както е показано в примера.

Това ще запази рецептите и придружаващите метаданни в локална папка.



Препроцесинг

```
with open('/app/data/epirecipes/full_format_recipes.json') as json_data:
    recipe_data = json.load(json_data)

filtered_data = [
    'Recipe for ' + x['title'] + ' | ' + ' '.join(x['directions'])
    for x in recipe_data
    if 'title' in x
    and x['title'] is not None
    and 'directions' in x
    and x['directions'] is not None
]
```

Примерът показва как данните могат да бъдат **заредени и филтрирани**, така че да останат само **рецепти със заглавие и описание**.

Пример за текстов низ на рецепта е даден в кода.



Примерна рецепта

Recipe for Ham Persillade with Mustard Potato Salad and Mashed Peas | Chop enough parsley leaves to measure 1 tablespoon; reserve. Chop remaining leaves and stems and simmer with broth and garlic in a small saucepan, covered, 5 minutes. Meanwhile, sprinkle gelatin over water in a medium bowl and let soften 1 minute. Strain broth through a fine-mesh sieve into bowl with gelatin and stir to dissolve. Season with salt and pepper. Set bowl in an ice bath and cool to room temperature, stirring. Toss ham with reserved parsley and divide among jars. Pour gelatin on top and chill until set, at least 1 hour. Whisk together mayonnaise, mustard, vinegar, 1/4 teaspoon salt, and 1/4 teaspoon pepper in a large bowl. Stir in celery, cornichons, and potatoes. Pulse peas with marjoram, oil, 1/2 teaspoon pepper, and 1/4 teaspoon salt in a food processor to a coarse mash. Layer peas, then potato salad, over ham.



Първо ще разгледаме ...

- » Преди да разгледаме как можем да изградим LSTM мрежа в Keras, първо ще разберем **структурата на текстовите данни**.
- » Също как тя се **различава от данните за изображения**, които видяхме досега в този лекционен курс.



Работа с текстови данни



Разлики

- » Има няколко ключови разлики между текстови и графични данни.
- » Това означава, че много от методите, които работят добре за графични данни, **не са лесно приложими** към текстови данни.



Обратно разпространение

- » Текстовите данни са съставени от **отделни части** (знаци или думи), докато пикселите в изображението са **точки в непрекъснат цвят** **спектр**.
 - > Лесно можем да направим зелен пиксел по-син, но не е очевидно как трябва да направим думата „котка“ по-подобна на думата „куче“ например.
- » С данните от едно изображение можем лесно да приложим обратно разпространение, понеже **можем да изчислим градиента** на функцията на загуба по отношение на отделните пиксели за да установим посоката, в която цветовете на пикселите трябва да се променят за да минимизираме загубата.
 - > С дискретни текстови данни очевидно **не можем да приложим обратно разпространение** по същия начин, така че трябва да намерим начин да заобиколим този проблем.



Времево и пространствено измерения

- » Текстовите данни **имат времево измерение, но нямат пространствено измерение.**
- » Графичните данни имат **две пространствени измерения, но нямат времево измерение.**
- » Редът на думите е много важен в текстовите данни и думите **не биха имали смисъл в обратен ред**, докато изображенията обикновено могат да бъдат обърнати без да се засяга съдържанието.
- » Освен това често има дългосрочни последователни зависимости между думите, които трябва да бъдат уловени от модела: например отговорът на въпрос или пренасянето на контекста на местоимение.
- » С данните за изображения всички пиксели могат да се обработват едновременно.



Чувствителност към промени

- » Текстовите данни са **силно чувствителни** към малки промени в отделните единици (думи или знаци).
- » Данните за изображения обикновено са **по-малко чувствителни към промените** в отделните пикселни единици (картина на къща все още ще бъде разпозната като къща, дори ако някои пиксели са били променени), но с текстови данни промяната дори на няколко думи може драстично да промени значението на пасажа или да го направи безсмислен.
- » Това прави **трудно обучението на модел да генерира съгласуван текст**, тъй като всяка дума е жизненоважна за цялостното значение на пасажа.



Структура, базирана на правила

- » Текстовите данни имат **базирана на правила граматична структура**, докато данните за изображения не следват определени правила за това как трябва да се присвояват стойностите на пикселите.
- » Има също **семантични правила**, които са изключително трудни за моделиране.



Напредък

- » Доскоро повечето от най-сложните генеративни дълбоки модели за обучение **се фокусираха върху данните за изображения**, тъй като много от предизвикателствата, представени в предходния списък, бяха извън обсега на дори и най-напредналите техники.
- » Въпреки това, през последните пет години беше постигнат удивителен напредък в областта на базираното на текст генериращо задълбочено обучение, благодарение на **въвеждането на архитектурата на трансформаторния модел**, който ще разгледаме в следващата лекция.



Обобщение

- » Имайки предвид тези предизвикателства, нека сега да разгледаме стъпките, които трябва да предприемем, за да **приведем текстовите данни в правилната форма за обучение на LSTM мрежа.**



Токенизация



Определение

- » Първата стъпка е да почистим и токенизираме текста.
- » Токенизирането е процес на **разделяне на текста на отделни единици**, като думи или знаци.
- » Начинът, по който токенизираме текста, ще зависи от това, което се опитваме да постигнем с модела за генериране на текст.
- » Има плюсове и минуси при използването както на думи, така и на знаци и изборът ще повлияе на начина, по който трябва да почистим текста преди моделиране, както и на изхода от модела.



Ако използваме токени с думи:

- » Целият текст може да се преобразува в **малки букви** за да се гарантира, че думите с главни букви в началото на изреченията **са токенизирани по същия начин**, както същите думи, появяващи се в средата на изречението.
- » В някои случаи обаче това може да не е желателно - например, някои собствени имена, като имена или места, може да се възползват от това да останат с главни букви, така че да бъдат токенизирани независимо.



Ако използваме токени с думи:

- » **Текстовият речник** (наборът от отделни думи в набора за обучение) може да е **много голям**, като някои думи се появяват **много рядко** или може би само веднъж.
- » Може да е разумно да заменим редките думи с **токен за неизвестна дума**, вместо да ги включваме като отделни токени
 - > За да намалим броя на теглата, които невронната мрежа трябва да научи.



Ако използваме токени с думи:

- » Думите могат да бъдат **преобразувани**, което означава, че те са **редуцирани до най-простата си форма**
 - > Така например различните времена на глагола остават токенизирани заедно.
 - > Ще трябва да токенизираме пунктуацията или да я премахнем напълно.
- » Използването на токенизиране на думи означава, че моделът никога няма да може да предвиди думи извън обучителния речник.



Ако използваме токени за знаци:

- » Моделът може да генерира поредици от знаци, които образуват **нови думи извън обучителния речник**
 - > Това може да е желателно в някои контексти, но не и в други.
- » Главните букви могат да бъдат преобразувани в техните двойници с малки букви или да останат като отделни символи.



Ако използваме токени за знаци:

- » Речникът обикновено е много по-малък, когато се използва символна токенизация.
- » Това е от полза за скоростта на обучение на модела, тъй като има по-малко тежести за изучаване в крайния изходен слой.



Пример

- » За този пример ще използваме токенизиране на думи с малки букви без корен на думата.
- » Ще токенизираме и препинателните знаци, тъй като бихме искали моделът да предвижда кога трябва да завършва изречения или да използва запетаи, например.



Почистване и токенизиране на текста

```
def pad_punctuation(s):  
    s = re.sub(f"([{{string.punctuation}}])", r' \1 ', s)  
    s = re.sub(' +', ' ', s)  
    return s  
  
text_data = [pad_punctuation(x) for x in filtered_data] ❶  
  
text_ds = tf.data.Dataset.from_tensor_slices(text_data).batch(32).shuffle(1000) ❷  
  
vectorize_layer = layers.TextVectorization( ❸  
    standardize = 'lower',  
    max_tokens = 10000,  
    output_mode = "int",  
    output_sequence_length = 200 + 1,  
)  
  
vectorize_layer.adapt(text_ds) ❹  
vocab = vectorize_layer.get_vocabulary() ❺
```

1. Препинателните знаци се третират като отделни думи.
2. Преобразуване в набор от данни TensorFlow.
3. Създаване слой Keras TextVectorization за преобразуване текст в малки букви:
 - ✓ Най-разпространените 10 000 думи
 - ✓ Съответен целочислен токен
 - ✓ Отрязване последователност до 201 токена.
4. Прилагане слоя TextVectorization към данните за обучение.
5. Променливата vocab съхранява списък с токени на думи.



Пример за рецепта след токенизиране

```
[ 26 16 557 1 8 298 335 189 4 1054 494 27 332 228
 235 262 5 594 11 133 22 311 2 332 45 262 4 671
 4 70 8 171 4 81 6 9 65 80 3 121 3 59
 12 2 299 3 88 650 20 39 6 9 29 21 4 67
 529 11 164 2 320 171 102 9 374 13 643 306 25 21
 8 650 4 42 5 931 2 63 8 24 4 33 2 114
 21 6 178 181 1245 4 60 5 140 112 3 48 2 117
```

```
557 8 285 235 4 200 292 980 2 107 650 28 72 4
108 10 114 3 57 204 11 172 2 73 110 482 3 298
 3 190 3 11 23 32 142 24 3 4 11 23 32 142
 33 6 9 30 21 2 42 6 353 3 3224 3 4 150
 2 437 494 8 1281 3 37 3 11 23 15 142 33 3
 4 11 23 32 142 24 6 9 291 188 5 9 412 572
 2 230 494 3 46 335 189 3 20 557 2 0 0 0
 0 0 0 0 0]
```

Дължината на последователността, която използваме за обучение на модела, е параметър на процеса на обучение.

В този пример избираме да използваме дължина на последователност 200, така че добавяме или изрязваме рецептата с една повече от тази дължина, за да ни позволи да създадем целевата променлива. За да се постигне тази желана дължина, край на вектора е подплатен с нули.

Стоп токени - токена 0 е известен като токен за спиране, което означава, че текстовият низ е достигнал своя край.



Пример за рецепта след токенизиране

```
0:  
1: [UNK]  
2: .  
3: ,  
4: and  
5: to  
6: in  
7: the  
8: with  
9: a
```

В примера можем да видим подмножество от списъка с токени, съпоставени към съответните им индекси.

Слоят запазва токена 0 за подпълване (т.е. това е маркерът за спиране) и токена 1 за неизвестни думи, които попадат извън първите 10 000 думи (напр. persillade).

Другите думи са присвоени токени по ред на честотата.

Броят на думите, които трябва да се включат в речника, също е параметър на процеса на обучение.

Колкото повече думи са включени, толкова по-малко неизвестни токени ще видите в текста - нашият модел обаче ще трябва да бъде по-голям, за да поеме по-големия размер на речника.



Следващата дума

- » Нашият LSTM ще бъде обучен да предсказва следващата дума в последователността.
 - > Например, можем да захраним модела с токени за „пиле на скара с варени ...“ и бихме очаквали моделът да изведе подходяща следваща дума (напр. „картофи“, а не „банани“).
- » Следователно можем просто да изместим цялата последователност с един токен за да създадем целевата променлива.



Стъпката за генериране на набор от данни може да бъде постигната с кода

```
def prepare_inputs(text):  
    text = tf.expand_dims(text, -1)  
    tokenized_sentences = vectorize_layer(text)  
    x = tokenized_sentences[:, :-1]  
  
    y = tokenized_sentences[:, 1:]  
    return x, y  
  
train_ds = text_ds.map(prepare_inputs) ❶
```

Създаване набора за обучение, състоящ се от токени за рецепта (входа) и същия вектор, изместен с един токен (целта).



LSTM Архитектура



Архитектурата на LSTM модел

Layer (type)	Output shape	Param #
InputLayer	(None, None)	0
Embedding	(None, None, 100)	1,000,000
LSTM	(None, None, 128)	117,248
Dense	(None, None, 10000)	1,290,000

Total params 2,407,248

Trainable params 2,407,248

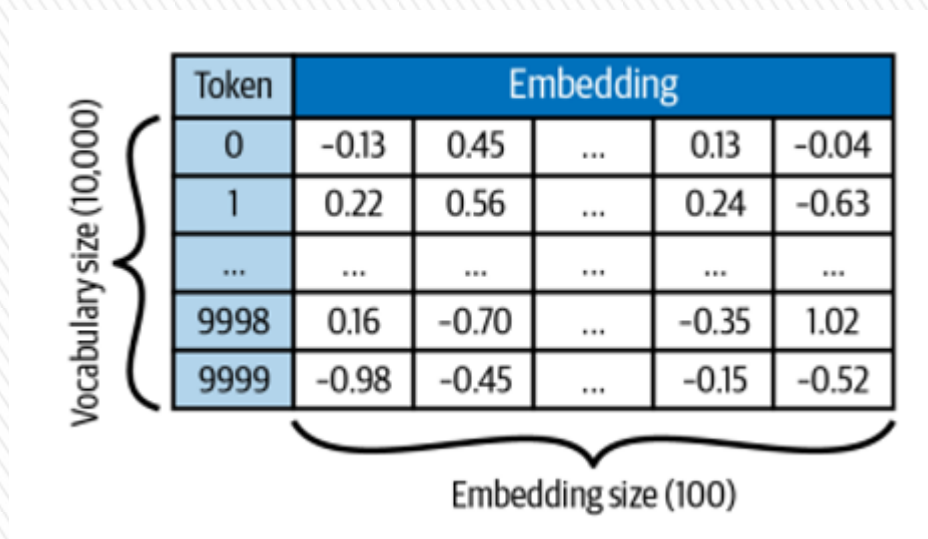
Non-trainable params 0

Входът към модела е **пореждане от цели числа**, а изходът е **вероятността всяка дума** в речника от 10 000 думи **да се появи като следваща** в поредицата.

За да разберем как работи това в детайли, трябва да представим два нови типа слоеве, **вграждане** и **LSTM**.



Слой за вграждане



The diagram illustrates an embedding layer as a table. On the left, a bracket labeled 'Vocabulary size (10,000)' spans the rows. The table has two main columns: 'Token' and 'Embedding'. The 'Embedding' column is further divided into five sub-columns. The rows represent individual tokens, with the first two being 0 and 1, followed by an ellipsis, and then 9998 and 9999. The 'Embedding' values are numerical vectors. A bracket at the bottom of the table is labeled 'Embedding size (100)', indicating the dimensionality of each vector.

Token	Embedding				
0	-0.13	0.45	...	0.13	-0.04
1	0.22	0.56	...	0.24	-0.63
...
9998	0.16	-0.70	...	-0.35	1.02
9999	-0.98	-0.45	...	-0.15	-0.52

Слойт за вграждане е по същество справочна таблица, която **преобразува всеки целочислен токен във вектор с дължина `embedding_size`**, както е показано на фигурата.

Векторите се научават от модела като **тегла**.

Следователно броят на теглата, научени от този слой, е равен на размера на речника, умножен по размерността на вектора за вграждане (т.е. $10\,000 \times 100 = 1\,000\,000$).

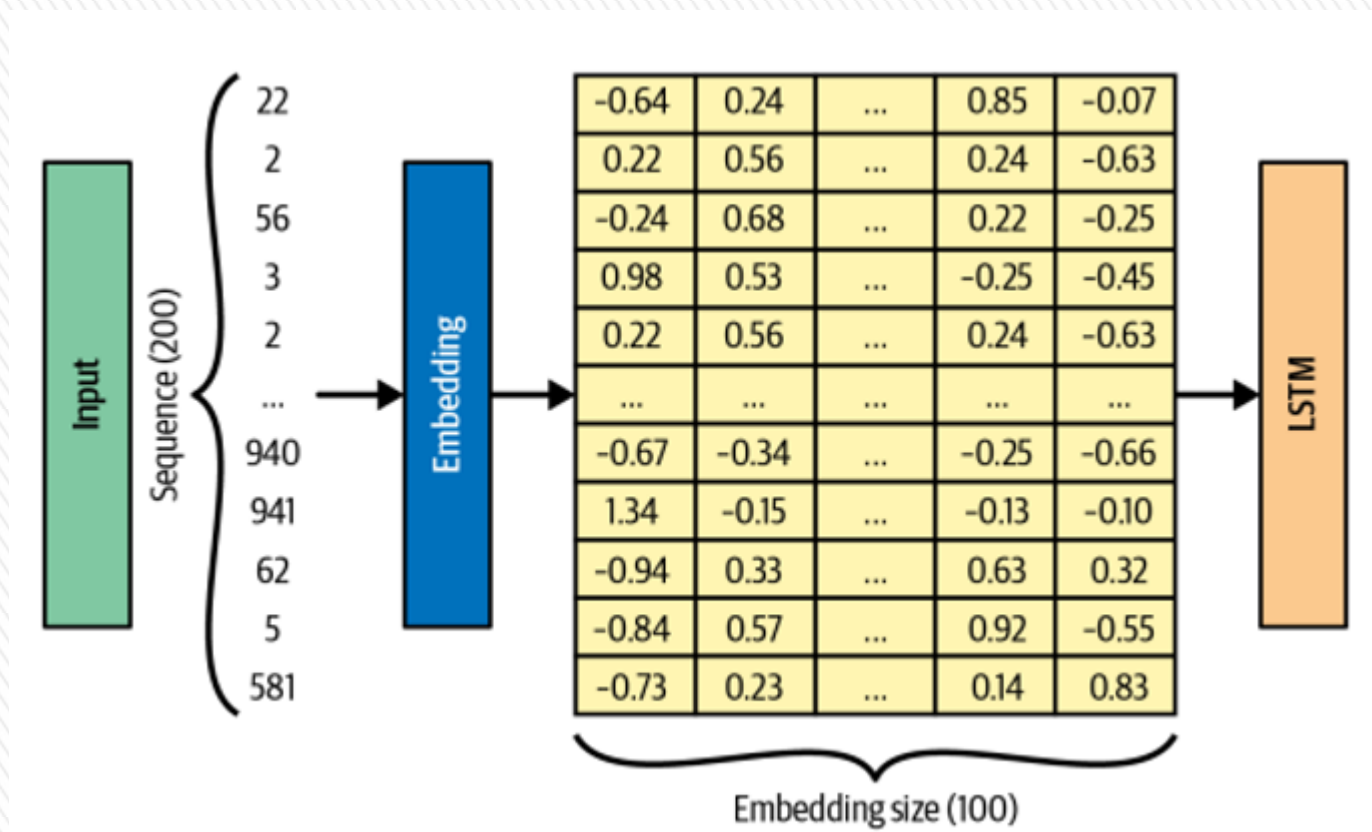


Слой за вграждане

- » Ние вграждаме всеки целочислен токен в непрекъснат вектор, защото позволява на модела да научи представяне за всяка дума, която може да се актуализира чрез обратно разпространение.
- » Можем също така просто да кодираме еднократно всеки входен токен, но използването на слой за вграждане е за предпочитане, защото **прави самото вграждане обучимо**, като по този начин дава на модела повече **гъвкавост** при вземането на решение как да вгради всеки токен за подобряване производителността.



Слой за вграждане



Следователно, входният слой предава **тензор от целочислени последователности** с форма $[\text{batch_size}, \text{seq_length}]$ към слоя за вграждане, който извежда **тензор с форма** $[\text{batch_size}, \text{seq_length}, \text{embedding_size}]$.

След това това се предава на слоя LSTM (фигурата).

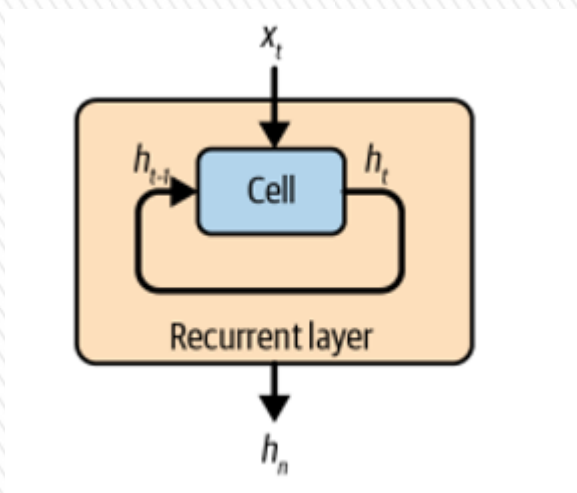


Слой LSTM

- » За да разберем слоя LSTM, първо трябва да разгледаме как работи един **общ повтарящ се слой**.
- » Повтарящият се слой има специалното свойство - може да обработва **последователни входни данни** x_1, \dots, x_n .
- » Състои се от клетка, която актуализира своето **скрито състояние** h_t когато отделните елементи от последователността x_t преминава постъпково във времето през нея.



Слой LSTM



Скритото състояние е вектор с дължина, равна на броя на единиците в клетката - може да се разглежда като текущото разбиране на клетката за последователността.

На времева стъпка t клетката използва предишната стойност на скритото състояние h_{t-1} заедно с данните от текущата времева стъпка x_t за създаване на актуализиран вектор на скрито състояние h_t .

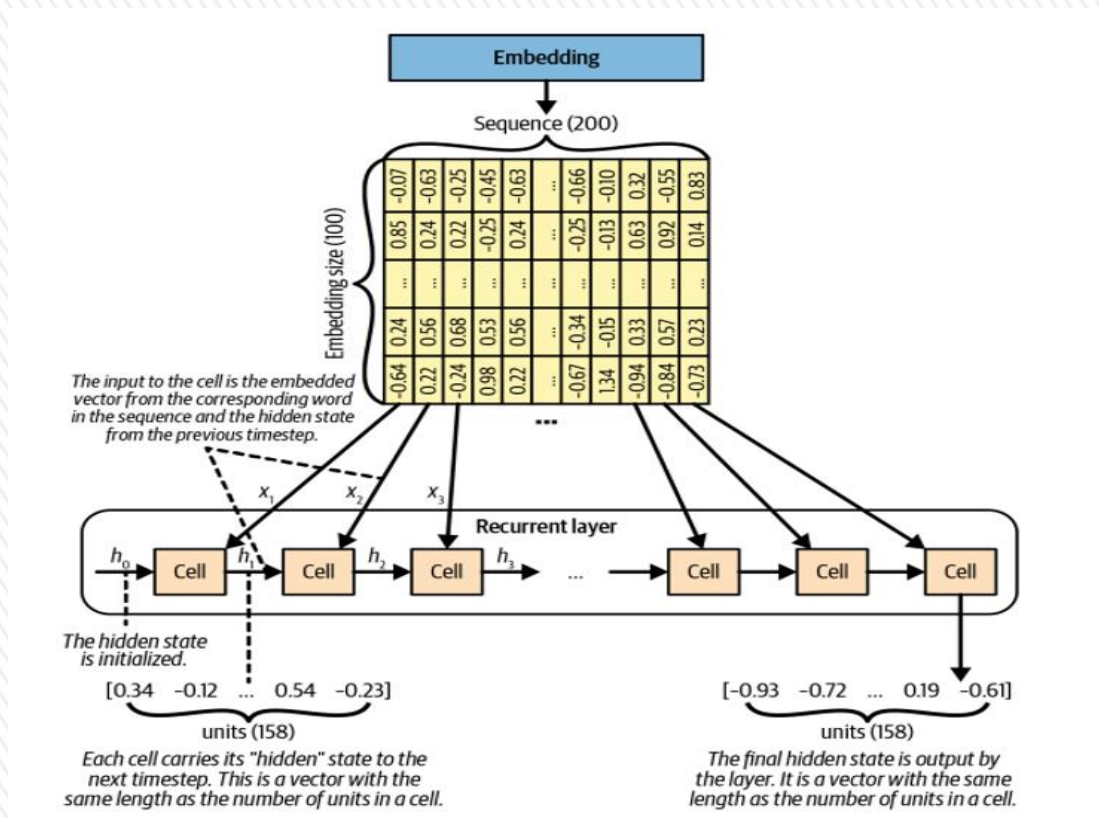
Този повтарящ се процес продължава до края на последователността.

След като последователността приключи, слойт извежда крайното скрито състояние на клетката h_n , което след това се предава на следващия слой на мрежата.

Този процес е показан на фигурата.



Слой LSTM



Детайли:

- ✓ Представяме повтарящия се процес, като задаваме копия на клетката на всяка времева стъпка
- ✓ Показваме как скритото състояние непрекъснато се актуализира, докато преминава през копията.
- ✓ Можем ясно да видим как предишното скрито състояние се смесва с текущата последователна точка от данни (т.е. текущият вграден вектор на думата) за да се получи следващото скрито състояние.
- ✓ Резултатът от слоя е **окончателното скрито състояние на клетката, след като всяка дума във входната последователност бъде обработена.**



LSTM клетка



Анатомия на LSTM клетка

- » Сега, след като видяхме как работи генеричен повтарящ се слой нека да погледнем **вътре в LSTM клетката**.
- » Задачата на клетката LSTM е да изведе ново **скрито състояние** h_t при зададено нейното предишно скрито състояние h_{t-1} и текущото вграждане на дума x_t .
 - > Дължината на h_t е **равна на броя на елементите** на LSTM.
- » Това е параметър, който се задава, когато дефинираме слоя
 - > Няма нищо общо с дължината на последователността.

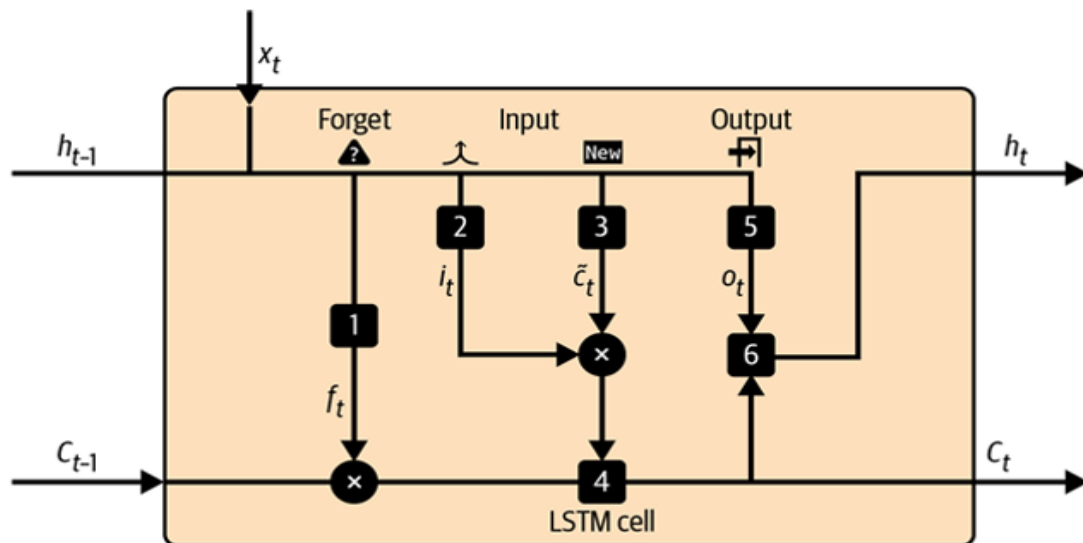


Анатомия на LSTM клетка

- » Една LSTM клетка поддържа състояние на клетката C_t , което може да се разглежда като **вътрешни вярвания** на клетката относно **текущото състояние на последователността**.
- » Това е **различно** от скритото състояние h_t , което в крайна сметка се извежда от клетката след последната времева стъпка.
- » Състоянието на клетката е със **същата дължина** като скритото състояние (броят единици в клетката).



Архитектура на LSTM клетка



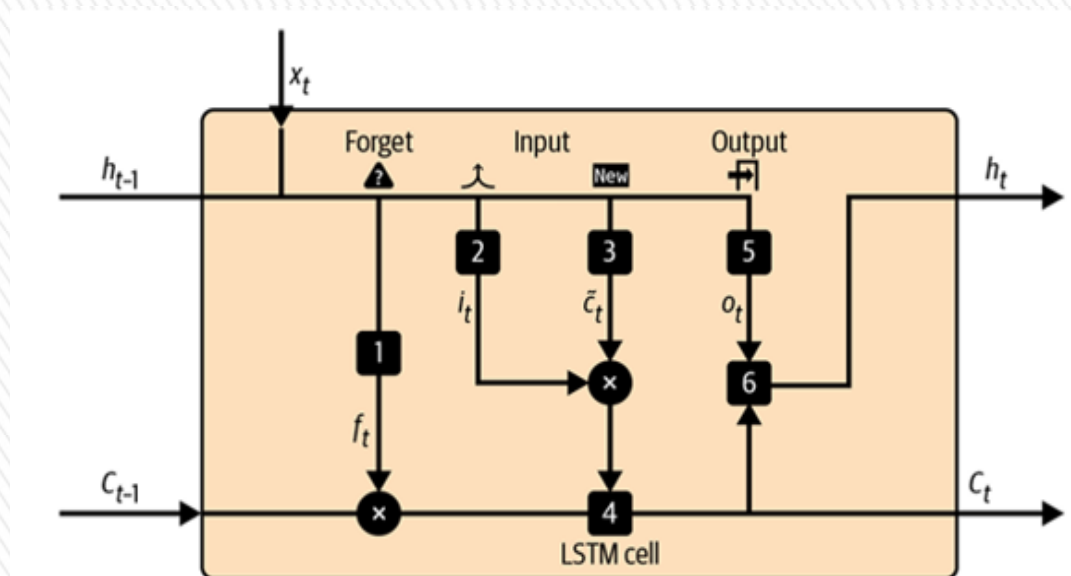
- 1 $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$
- 2 $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$
- 3 $\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$
- 4 $C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$
- 5 $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$
- 6 $h_t = o_t \cdot \tanh(C_t)$

Нека разгледаме по-подробно една клетка и как се актуализира скритото състояние (фигурата).

Скритото състояние се актуализира в шест стъпки.



Архитектура на LSTM клетка

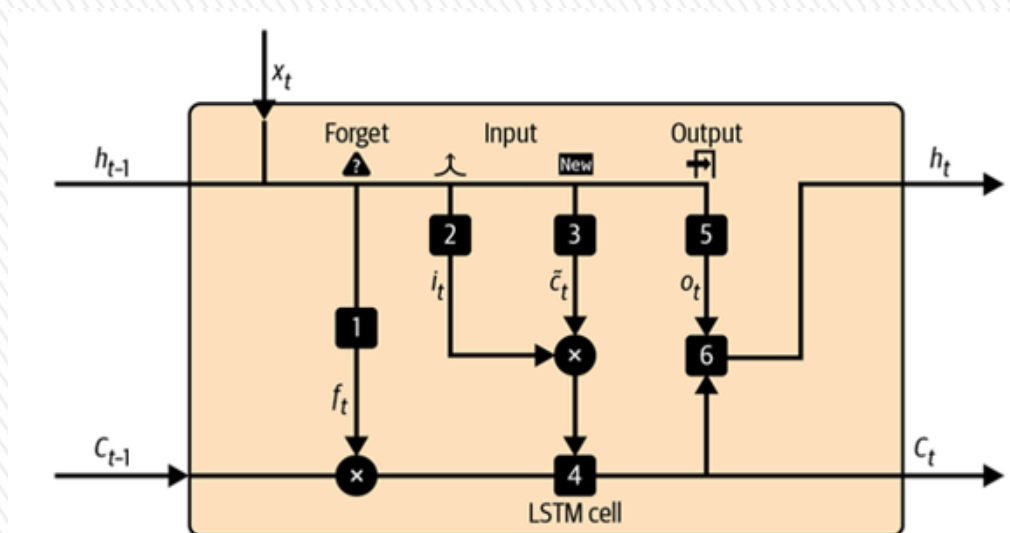


1 $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$

1. Скритото състояние на предишната времева стъпка h_{t-1} и текущата вградена дума x_t се свързват и преминават през **порта за забравяне**.
 - ✓ Този порт е просто **плътен слой** с матрица на теглата W_f , отклонение b_f и сигмоидна функция за активиране.
 - ✓ Полученият вектор f_t има дължина, равна на броя елементите на клетката и съдържа стойности между 0 и 1, които определят **каква част от предишното състояние на клетката c_{t-1} трябва да се запази**.



Архитектура на LSTM клетка



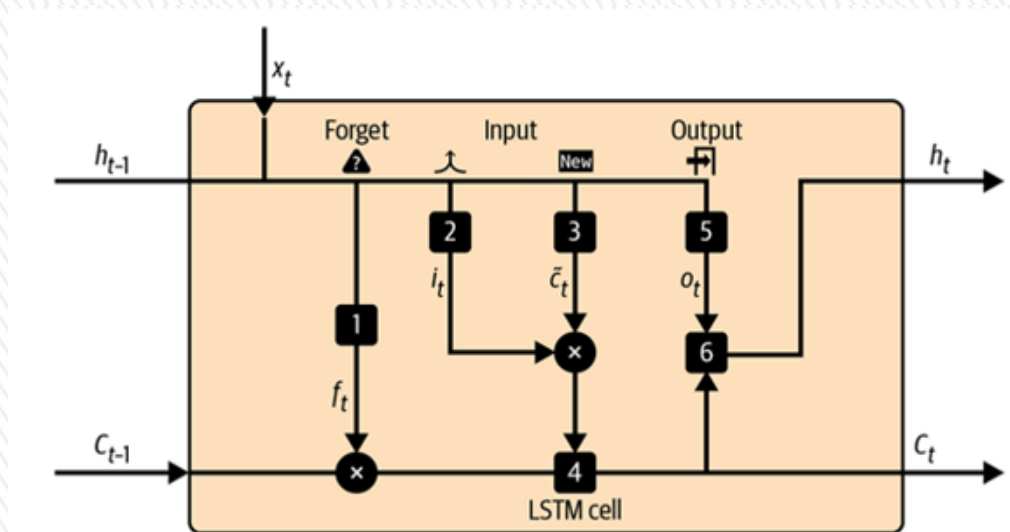
2 $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$

2. Конкатенираният вектор също се прекарва през входен порт, който, подобно на порта за забравяне е **плътен слой** с матрица на теглата W , отклонение b_i и сигмоидна функция за активиране.

- ✓ Изходът от този порт има дължина, равна на броя на единиците в клетката и съдържа стойности между 0 и 1, които определят колко нова информация ще бъде добавена към предишното състояние на клетката C_{t-1} .



Архитектура на LSTM клетка



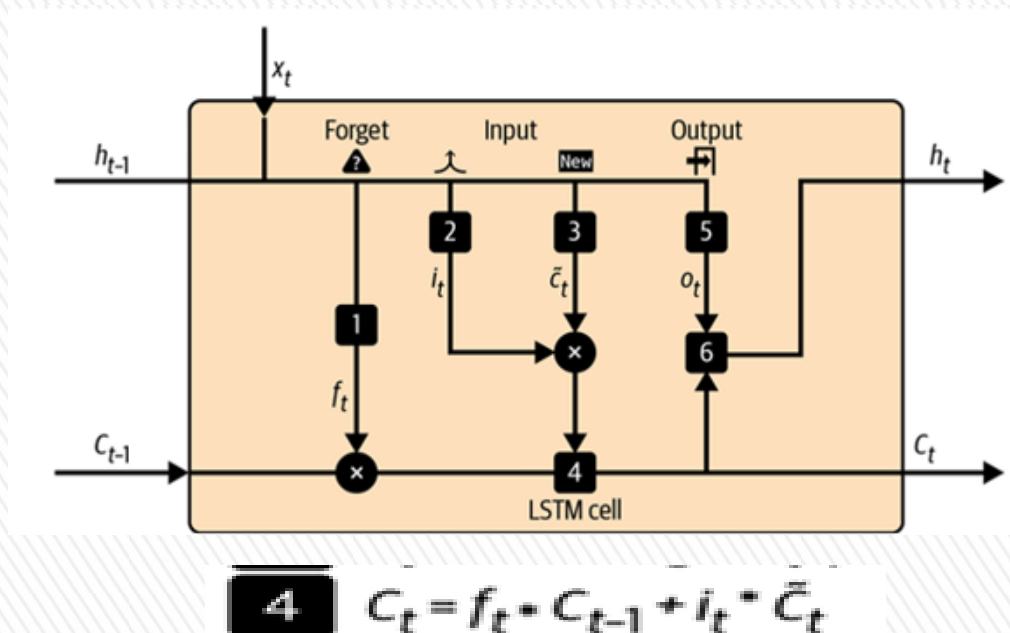
3 $\tilde{c}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$

3. Конкатенираният вектор се прекарва през плътен слой с матрица на теглата W_C , отклонение b_C и една \tanh активираща функция за да генерира един вектор \tilde{c}_t , който съдържа нова информация че клетката иска да обмисли запазването.

✓ Освен това има дължина, равна на броя на единиците в клетката, и съдържа стойности между -1 и 1 .



Архитектура на LSTM клетка

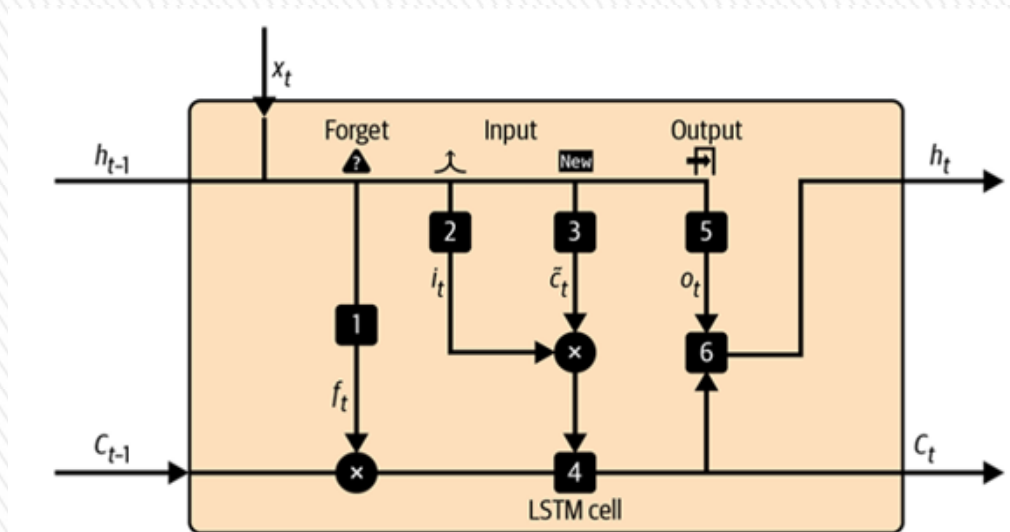


4. f_t и C_{t-1} се умножават поелементно и се добавят към поелементното умножение на i_t и \tilde{C}_t

✓ Това представлява забравяне на части от предишното състояние на клетката и след това добавяне на нова подходяща информация за създаване на актуализираното състояние на клетката C_t



Архитектура на LSTM клетка



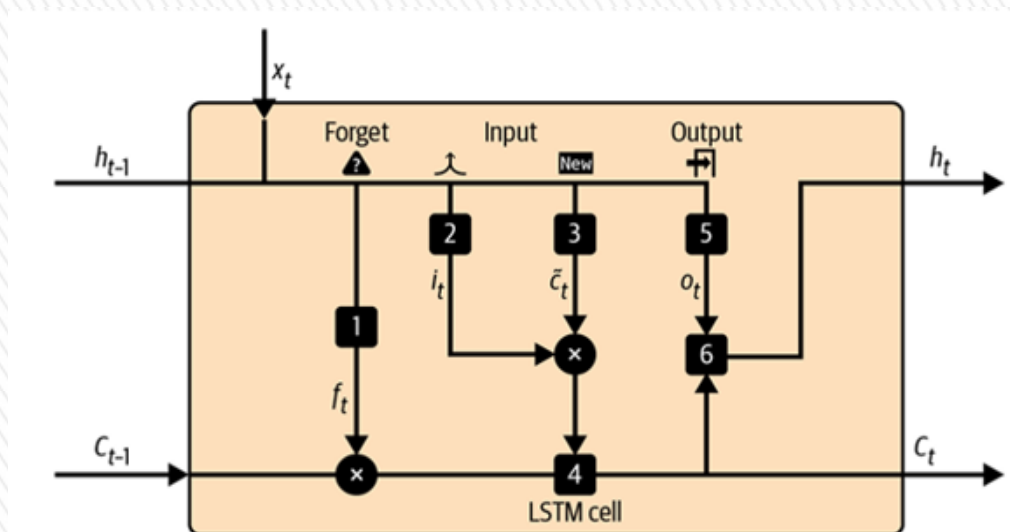
5
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

5. Конкатенираният вектор се прекарва през изходящ гейт: плътен слой с матрица на теглата W_o , отклонение b_o и едно сигмоидно активиране.

✓ Резултиращият вектор o_t има дължина равна на броя на единиците в клетката и съхранява стойности между 0 и 1, които определят каква част от актуализираното състояние на клетката C_t да извеждат от клетката.



Архитектура на LSTM клетка



6. o_t се умножава по елементи с актуализираното състояние на клетката C_t , след което едно \tanh активиране се прилага за да произведе новото скрито състояние h_t .

$$6 \quad h_t = o_t \cdot \tanh(C_t)$$



Обобщение

- » Цялата тази сложност е обвита в LSTM слоя на Keras.
- » Така че не е нужно да се притеснявам да го внедряваме сами.



Код в Keras

```
inputs = layers.Input(shape=(None,), dtype="int32") ❶  
x = layers.Embedding(10000, 100)(inputs) ❷  
x = layers.LSTM(128, return_sequences=True)(x) ❸  
outputs = layers.Dense(10000, activation = 'softmax')(x) ❹  
lstm = models.Model(inputs, outputs) ❺  
  
loss_fn = losses.SparseCategoricalCrossentropy()  
lstm.compile("adam", loss_fn) ❻  
lstm.fit(train_ds, epochs=25) ❼
```

1. Входният слой не се нуждае от предварително указване на дължината на последователността (може да бъде гъвкав), така че използваме None като контейнер.
2. Слой за вграждане изисква два параметъра, размера на речника (10 000 токена) и размерността на вектора за вграждане (100).
3. Слоеве LSTM изискват от нас да посочим размерността на скрития вектор (128). Също така избираме да върнем пълната последователност от скрити състояния, а не само скритото състояние на последния времеви етап.



Код в Keras

```
inputs = layers.Input(shape=(None,), dtype="int32") ❶
x = layers.Embedding(10000, 100)(inputs) ❷
x = layers.LSTM(128, return_sequences=True)(x) ❸
outputs = layers.Dense(10000, activation = 'softmax')(x) ❹
lstm = models.Model(inputs, outputs) ❺

loss_fn = losses.SparseCategoricalCrossentropy()
lstm.compile("adam", loss_fn) ❻
lstm.fit(train_ds, epochs=25) ❼
```

4. Плътният слой трансформира скритите състояния на всяка времева стъпка във вектор от вероятности за следващия токен.
5. Цялостният модел предвижда следващия токен, дадена входна последователност от токени. Той прави това за всеки токен в последователността.
6. Моделът е компилиран със загуба на SparseCategoricalCrossentropy—това е същото като категоричната кръстосана ентропия, но се използва, когато етикетите са цели числа, а не еднократно кодирани вектори.
7. Моделът е подходящ за набора от данни за обучение.



Първи няколко епохи от процеса на обучение на LSTM

```
Epoch 1/25
628/629 [=====>.] - ETA: 0s - loss: 4.4536
generated text:
recipe for mold salad are high 8 pickled to fold cook the dish into and warm in baking reduced but halves beans
and cut

629/629 [=====] - 29s 43ms/step - loss: 4.4527
Epoch 2/25
628/629 [=====>.] - ETA: 0s - loss: 3.2339
generated text:
recipe for racks - up-don with herb fizz | serve checking thighs onto sanding butter and baking surface in a hea
vy heavy large saucepan over blender ; stand overnight . [UNK] over moderate heat until very blended , garlic ,
about 8 minutes . cook airtight until cooked are soft seeds , about 1 45 minutes . sugar , until s is brown , 5
to sliced , parmesan , until browned and add extract . wooden crumb to outside of out sheets . flatten and prehe
ated return to the paste . add in pecans oval and let transfer day .

629/629 [=====] - 30s 48ms/step - loss: 3.2336
Epoch 3/25
629/629 [=====] - ETA: 0s - loss: 2.6229
generated text:
recipe for grilled chicken | preheat oven to 400*f . cook in large 8 - caramel grinder or until desired are firm
, about 6 minutes

629/629 [=====] - 27s 42ms/step - loss: 2.6229
Epoch 4/25
629/629 [=====] - ETA: 0s - loss: 2.3426
generated text:
recipe for pizza salad with sweet red pepper and star fruit | combine all ingredients except lowest ingredients
in a large skillet . working with batches and deglaze , cook until just cooked through , about 1 minute . meanwh
ile , boil potatoes and paprika in a little oil over medium - high heat , stirring it just until crisp , about 3
minutes . stir in bell pepper , onion and cooked paste and jalapeño until clams well after most - reggiano , abo
ut 5 minutes . transfer warm 2 tablespoons flesh of eggplants to medium bowl . serve .
```

Забележете как примерният изход става по-разбираем, когато показателят за загуба пада.



Метрика за загуба на кръстосана ентропия



Анализ на LSTM



Използване на LSTM

- » Сега, след като компилирахме и обучихме LSTM, можем да започнем да я използваме за генериране на дълги низове от текст, като приложим следния процес:
 - > Захранваме мрежата със съществуваща последователност от думи и искаме да предвиди следващата дума.
 - > Добавяме тази дума към съществуващата последователност и повтаряме процеса.



Използване на LSTM

- » Мрежата ще изведе набор от вероятности за всяка дума, от която можем да направим извадка.
- » Следователно можем да направим генерирането на текст стохастично, а не детерминистично.
- » Освен това можем да въведем температурен параметър в процеса на вземане на проби, за да посочим колко детерминистичен бихме искали да бъде процесът.



Температурен параметър

- » Температура, близка до 0:
 - > **Прави извадката по-детерминистична** - т.е. думата с най-голяма вероятност е много вероятно да бъде избрана
- » Температура 1:
 - > Означава, че **всяка дума е избрана с изхода на вероятността от модела.**
- » Това се постига с кода на следващия слайд, който създава функция за обратно извикване, която може да се използва за генериране на текст в края на всяка епоха на обучение.



Код на Keras

```
class TextGenerator(callbacks.Callback):
    def __init__(self, index_to_word, top_k=10):
        self.index_to_word = index_to_word
        self.word_to_index = {
            word: index for index, word in enumerate(index_to_word)
        } ❶

    def sample_from(self, probs, temperature): ❷
        probs = probs ** (1 / temperature)
        probs = probs / np.sum(probs)
        return np.random.choice(len(probs), p=probs), probs

    def generate(self, start_prompt, max_tokens, temperature):
        start_tokens = [
            self.word_to_index.get(x, 1) for x in start_prompt.split()
        ] ❸
        sample_token = None
        info = []
        while len(start_tokens) < max_tokens and sample_token != 0: ❹
            x = np.array([start_tokens])
            y = self.model.predict(x) ❺
            sample_token, probs = self.sample_from(y[0][-1], temperature) ❻
            info.append({'prompt': start_prompt, 'word_probs': probs})
            start_tokens.append(sample_token) ❼
            start_prompt = start_prompt + ' ' + self.index_to_word[sample_token]
        print(f"\ngenerated text:\n{start_prompt}\n")
        return info

    def on_epoch_end(self, epoch, logs=None):
        self.generate("recipe for", max_tokens = 100, temperature = 1.0)
```

1. Създайте обратно картографиране на речника (от дума към токен).
2. Тази функция актуализира вероятностите с коефициент на мащабиране на температурата.
3. Подканата за стартиране е низ от думи, които бихте искали да дадете на модела, за да започне процеса на генериране (например рецепта за). Думите първо се преобразуват в списък от токени.
4. Последователността се генерира, докато стане max_tokens дълга или се генерира стоп токен (0).



Код на Keras

```
class TextGenerator(callbacks.Callback):
    def __init__(self, index_to_word, top_k=10):
        self.index_to_word = index_to_word
        self.word_to_index = {
            word: index for index, word in enumerate(index_to_word)
        } ❶

    def sample_from(self, probs, temperature): ❷
        probs = probs ** (1 / temperature)
        probs = probs / np.sum(probs)
        return np.random.choice(len(probs), p=probs), probs

    def generate(self, start_prompt, max_tokens, temperature):
        start_tokens = [
            self.word_to_index.get(x, 1) for x in start_prompt.split()
        ] ❸
        sample_token = None
        info = []
        while len(start_tokens) < max_tokens and sample_token != 0: ❹
            x = np.array([start_tokens])
            y = self.model.predict(x) ❺
            sample_token, probs = self.sample_from(y[0][-1], temperature) ❻
            info.append({'prompt': start_prompt, 'word_probs': probs})
            start_tokens.append(sample_token) ❼
            start_prompt = start_prompt + ' ' + self.index_to_word[sample_token]
        print(f"\ngenerated text:\n{start_prompt}\n")
        return info

    def on_epoch_end(self, epoch, logs=None):
        self.generate("recipe for", max_tokens = 100, temperature = 1.0)
```

5. Моделът извежда вероятностите всяка дума да е следваща в последователността.
6. Вероятностите преминават през семплера, за да се изведе следващата дума, параметризирана по температура.
7. Добавяме новата дума към подканващия текст, готови за следващата итерация на процеса на генериране.



Резултат

temperature = 1.0

generated text:

recipe for sour japanese potatoes julienne | in a bowl stir together the yeast mixture with the milk and the peanut butter crumbs , the sour cream , and the butter mixture with a fork , gently fold in the prunes gently or until incorporated . lightly stir the oil and yeast until it just holds soft peaks , but not runny , on bottom of a 7 - sided sheet of aluminum foil , top it with a round , and a pinch of each brownies into a goblet , or with the baking dish . serve each with sorbet

temperature = 0.2

generated text:

recipe for grilled chicken with mustard - herb sauce | combine first 6 ingredients in medium bowl . add chicken to pot . add chicken and turn to coat . cover and refrigerate at least 1 hour and up to 1 day . preheat oven to 450°f . place turkey on rack in roasting pan . roast until thermometer inserted into thickest part of thigh registers 175°f , about 1 hour longer . transfer to rack in center of oven and preheat to 450°f . brush chicken with oil . sprinkle with salt and pepper . roast until thermometer inserted into

Има няколко неща, които трябва да се отбележат относно тези два пасажа.

Първо, и двете са стилистично подобни на рецепта от оригиналния тренировъчен комплект.

И двете започват със заглавие на рецепта и съдържат като цяло граматически правилни конструкции.

Разликата е, че генерираният текст с температура 1.0 е по-приключенски и следователно по-малко точен от примера с температура 0.2.

Следователно генерирането на множество проби с температура 1,0 ще доведе до повече разнообразие, тъй като моделът взема проби от вероятно разпределение с по-голяма дисперсия.



Първи токени

temperature = 1.0	
PROMPT: recipe for roast	
turkey:	22.81%
chicken:	19.41%
beef:	10.24%
pork:	9.96%
leg:	4.06%

PROMPT: recipe for roasted vegetables	
preheat:	69.63%
prepare:	3.68%
heat:	3.45%
put:	2.12%
combine:	1.96%

PROMPT: recipe for chocolate ice cream	
in:	27.31%
combine:	11.21%
stir:	6.66%
whisk:	5.64%
mix:	3.68%

PROMPT: recipe for roasted vegetables chop 1 /	
2:	53.51%
4:	29.83%
3:	13.11%
8:	0.78%
1:	0.56%

temperature = 0.2	
PROMPT: recipe for roast	
turkey:	67.54%
chicken:	30.15%
beef:	1.23%
pork:	1.07%
leg:	0.01%

PROMPT: recipe for roasted vegetables	
preheat:	100.0%
prepare:	0.0%
heat:	0.0%
put:	0.0%
combine:	0.0%

PROMPT: recipe for chocolate ice cream	
in:	98.71%
combine:	1.15%
stir:	0.09%
whisk:	0.04%
mix:	0.0%

PROMPT: recipe for roasted vegetables chop 1 /	
2:	94.81%
4:	5.11%
3:	0.08%
8:	0.0%
1:	0.0%

За да демонстрира това фигурата показва първите токена с най-високи вероятности за набор от подкани и за двете температурни стойности.



Обобщение

- » Моделът е в състояние да генерира подходящо разпределение за следващата най-вероятна дума в редица контексти.
- » Например, въпреки че на модела никога не е било казано за части на речта като съществителни, глаголи или числа, той обикновено е в състояние да раздели думите в тези класове и да ги използва по начин, който е граматически правилен.



Обобщение

- » Освен това моделът може да избере подходящ глагол, за да започне инструкциите за рецептата, в зависимост от предходното заглавие.
- » За печени зеленчуци той избира предварително загревяване, приготвяне, загревяване, поставяне или комбиниране като най-вероятните възможности, докато за сладолед избира включване, комбиниране, разбъркване, разбиване и смесване.
- » Това показва, че моделът има известно контекстуално разбиране на разликите между рецептите в зависимост от техните съставки.



Обобщение

- » Забележете също така как вероятностите за температурата = 0,2 примери са много по-тежко претеглени към жетона на първия избор.
- » Това е причината, поради която обикновено има по-малко разнообразие в поколенията, когато температурата е по-ниска.



Обобщение

- » Докато основният ни модел LSTM върши страхотна работа при генерирането на реалистичен текст, ясно е, че той все още се бори да схване част от семантичното значение на думите, които генерира.
- » Той въвежда съставки, които няма вероятност да работят добре заедно (например кисели японски картофи, трохи от пекан и сорбе)!
- » В някои случаи това може да е желателно – да речем, ако искаме нашият LSTM да генерира интересни и уникални модели от думи – но в други случаи ще се нуждаем от нашия модел, за да имаме по-задълбочено разбиране на начините, по които думите могат да бъдат групирани заедно и по-дълга памет на идеи, въведени по-рано в текста.



Благодаря за вниманието!

