Търсене и извличане на информация.

Приложение на дълбоко машинно обучение

Зимен семестър 2020/2021

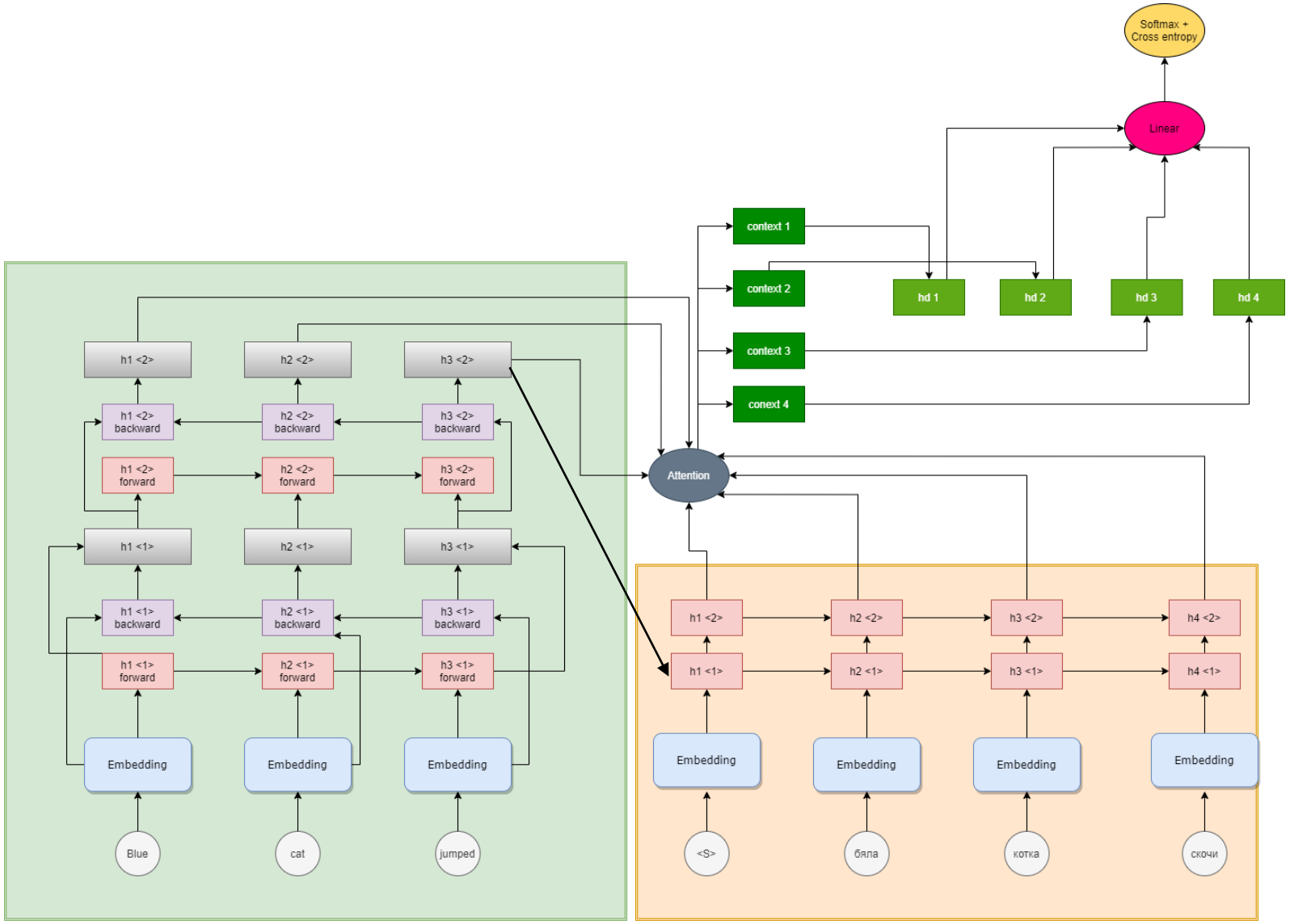
Курсов проект

Невронен машинен превод

Име: Любомир Георгиев Иванов

Факултетен номер: 62333

1. Диаграма на модела



2. Описание на решението

Модела е изграден от две подмрежи.

1. **Encoder** мрежата (в големия светлосин правоъгълник).

Тя представлява двуслойна, двупосочна LSTM рекурентна невронна мрежа.

2. Втората мрежа е **decoder** мрежата

(в големия светлочервен правоъгълник).

Тя представлява двуслойна, еднопосочна LSTM рекурентна невронна мрежа.

Контекстните вектори от последния слой на encoder мрежата и контекстните вектори от последния слой на decoder мрежата преминават през т.нар Attention слой, след което получаваме векторите за внимание за всяка стъпка от процеса по декодиране на входното изречение.

Тези вектори се конкатенират с контекстните вектори от последния слой на decoder мрежата, преминават през линеен слой, дропаут слой и след това се изчислява съответната крос-ентропия.

Дропаут слоеве има и след първите слоеве на decoder и encoder мрежите, но реших да не ги визуализирам за по-лесно онагледяване на схемата на модела.

Векторът за внимание е добавен след рекуректния слой, реализирайки вниманието чрез late binding.

3. Реализация на внимание

Първо ще онагледим някои размерности:

outputEncoder: (enc_seq, batch_size, 2 * hidden_size)

outputDecoder: (dec_seq, batch_size, 2 * hidden_size)

Ua -> (2 * hidden_size, hidden_size)

Wa -> (2 * hidden_size, hidden_size)

V -> (hidden_size, 1)

projection -> (4 * hidden_size, len(targetWord2Ind)), където

hidden_size е размера на скрития вектор на encoder мрежата, enc_seq е дължината на входната последователност (на английски), а dec_seq е дължината на изходната последователност (на български)

За да се възползваме максимално от скоростта на действията с векторни операции, ще се опитаме да изчислим всичките контекстни вектори за внимание наведнъж.

```
e = self.V(self.tanh(self.Ua(outputEncoder) + self.Wa(outputDecoder.unsqueeze(1))))
attentionWeights = self.softmaxForward(e)
context = torch.sum(attentionWeights * outputEncoder, dim=1)
a = torch.cat((context, outputDecoder), -1)
```

Следният код постига точно това.

Използваната функцията за внимание е т.нар Bahdanau attention:

$$e_{ij} = v^T \tanh(W s_{i-1} + V h_j)$$

Функцията е подробно описана в статията:

[Neural Machine Translation by Jointly Learning to Align and Translate]

Горната формула е за получаването на векторите за внимание за един encoder и един decoder вектор.

В нашия случай матрицата W е матрицата Wa,

матрицата V е матрицата Ua,

hj е един от векторите в outputEncoder, s е един от векторите в outputDecoder.

Нека разгледаме отделните равенства.

self.Ua(outputEncoder) реализира линейна трансформация на контекстните вектори, получени от encoder мрежата.

Аналогично self.Wa(outputDecoder) реализира линейна трансформация на контекстните вектори, получени от decoder мрежата, като използваме фунцкията unsqueeze(), за да се възползваме от broadcasting механизма на python и при сумата на двете трансформации да получим резултат, който е във форма (dec_seq, enc_seq, batch_size, hidden_size)

След това се прилага функцията tanh и още една линейна трансформация, след която размерността става (dec_seq, enc_seq, batch_size, 1)

Прилагаме softmax слой и числата за всеки encoder hidden state се преобразуват до вероятности, които се сумират до единица.

Следва умножение на получения резултат с outputEncoder, за да получим преобразуваните контекстни вектори на encoder мрежата и сумираме по първото второто измерение (dim=1), за да получим резултатния контекстен вектор за внимание.

Този вектор е с размерност (dec_seq, batch_size, 2 * hidden_size)

Накрая конкатенираме вектора за внимание с outputDecoder и получаваме `a = torch.cat((context, outputDecoder), -1)`, който е с размерност

dec_seq, batch_size, 4 * hidden_size.

За всяка дума от изходната последователност и за всички изречения в съответния batch получаваме вектор с размерност 4 * hidden_size, който ще използваме за изчисляване на съответната перплексия и генериране на нова дума при извършване на превода.

За осъществяването на превода на изречения е използван Greedy search.

4. Параметри за обучение

Параметрите за обучение са:

Learning rate: 0.003

Learning rate decay: 0.5

Uniform init: 0.1

Batch size: 32

Embed size: 128

Hidden size: 128

Dropout: 0.2

Linear dropout: 0.4

В началото на обучението, модела започва да се тренира с learning rate равен на 0.003. След лична преценка, че модела започва да се тренира много бавно (3 или 4 епохи перплексията не спада), първо learning rate се смъква до 0.002, след това 0.001, като най-ниската използвана стойност е 0.0005 за около 2 епохи, но подобренията постигнати с нея бяха пренебрежимо ниски и обучението може да бъде прекратено още при стойността 0.001.

Използван е оптимизатор Adam с настройките по подразбиране.

5. Среда за разработка и трениране

За разработката и тренирането на модела е използвана средата Google Colab, която е конфигурирана за работа с графична карта (GPU).

6. Резултати от тренирането

Получените крайни стойности за следените метрики са, както следва:

1) Перплексия за валидиращия корпус dev.en – **4.43**

2) Перплексия за тестовия корпус test.en – **4.74**

3) Bleu score за тестовия корпус test.en – **36.42**

7. Литература

[Bahdanau et al.(2015)Bahdanau, Cho, and Bengio] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. http://arxiv.org/abs/1409.0473.