Машинен превод от български към английски език

Име – Бойко Красимиров Борисов

Факултетен номер – 82033

Обученият модел достига следните резултати върху тестовото множество:

- Перплексия – 5,39
- BLEU оценка – 40,71

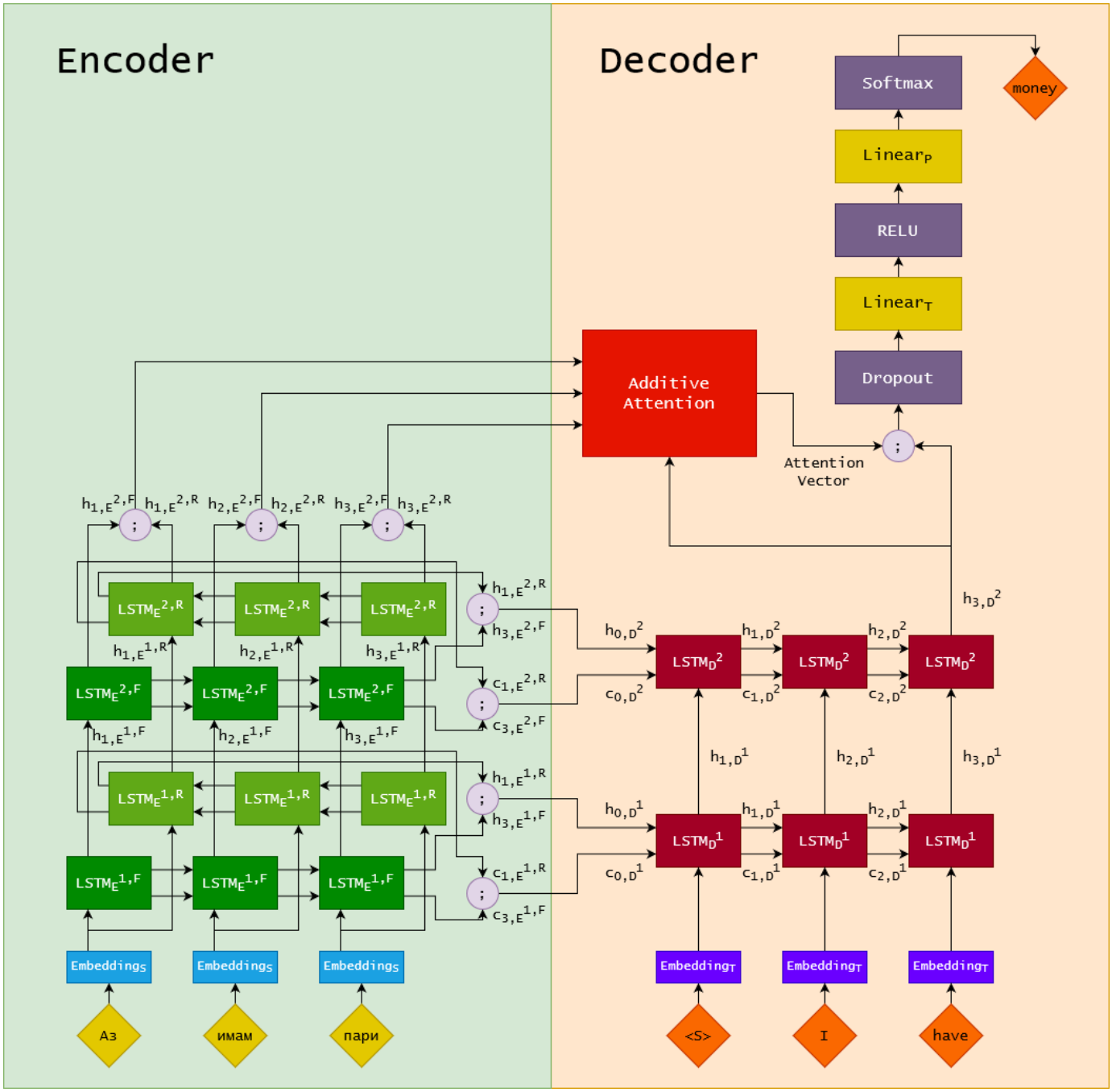
Архитектура на модела

Моделът има Encoder-Decoder структура като Encoder частта е реализирана чрез многослоен Bi-LSTM, докато Decoder частта е реализирана чрез многослоен LSTM с допълнителен модул за внимание. По същество както Encoder, така и Decoder частта представляват рекурентни мрежи.

По-долу е представена цялостната архитектура на модела. Със символа **“;”** е обозначената операцията конкатенация на вектори. Архитектурата е съставена от следните компоненти:

- Входни думи за езика източник (български) и езика цел (английски), които може да бъдат произволно на брой.
- Модули $\textbf{Embedding}_{\textbf{S}}$ и $\textbf{Embedding}_{\textbf{T}}$ за влагане на думи, които влагат думите съответно от езика източник и езика цел. Като резултат дават вектори с размерност **wordEmbeddingSize**.
- Двупосочен многослоен LSTM, реализиращ Encoder частта. В правата посока LSTM компонентът е именуван $\textbf{LSTM}_{\textbf{E}}^{\textbf{F}}$, докато в обратната $\textbf{LSTM}_{\textbf{E}}^{\textbf{R}}$, като допълнително в горния индекс е указан и номера на слоя. Броят на слоевете е **encoderLayers** (на схемата са показани само два слоя, но може да са произволен брой). Скритите вектори имат размерност **encoderHiddenSize**, като първият слой приема вход с размерност **wordEmbeddingSize**, докато следващите слоеве приемат изходите от предходния слой, чиято размерност е **encoderHiddenSize**.
- Еднопосочен многослоен LSTM, реализиращ част от Decoder частта. Той е именуван с $\textbf{LSTM}_{\textbf{D}}$, като допълнително в горния индекс е указан номера на слоя. Броят на слоевете е **decoderLayers** (на схемата са показани само два слоя). Скритите вектори имат размерност **decoderHiddenSize**, като първият слой приема вход с размерност **wordEmbeddingSize**, докато следващите слоеве приемат изходите от предходния слой, чиято размерност е **decoderHiddenSize**. Като първоначални стойности за скритите вектори и скритите състояния се подават директно резултати от Encoder Bi-LSTM-а, поради което трябва да са изпълнени равенствата **“decoderHiddenSize = 2 * encoderHiddenSize”** и **“decoderLayers = encoderLayers”**.
- Компонент за внимание **Additive Attention**, който е част от декодера. Той приема произволен брой контекстни вектори от последния слой на енкодера (на схемата са показани само 3, но може да са произволен брой), чиято размерност е **2 * encoderHiddenSize**. Освен това приема и един скрит вектор от декодера с размерност **decoderHiddenSize**. Той има и скрита размерност **attentionHiddenSize**, която използва в реализацията (за повече информация вижте описанието на реализацията на компонента по-долу)

- Компонент за регуларизация **Dropout**, който извършва отпадане с коефициент **dropout**.
- Линейна трансформация **Linear$_T$**. Тя приема вектор с размерност **2 * encoderHiddenSize + decoderHiddenSize** и връща вектор с размерност **projectionTransformSize**.
- **RELU** компонент за внасяне на нелинейност, чрез прекарване през ReLU функцията.
- Линейна трансформация **Linear$_P$**. Тя приема вектор с размерност **projectionTransformSize** и връща вектор с размерност броя налични думи за целевия език (английски).
- **Softmax** за нормализиране на вероятностите.

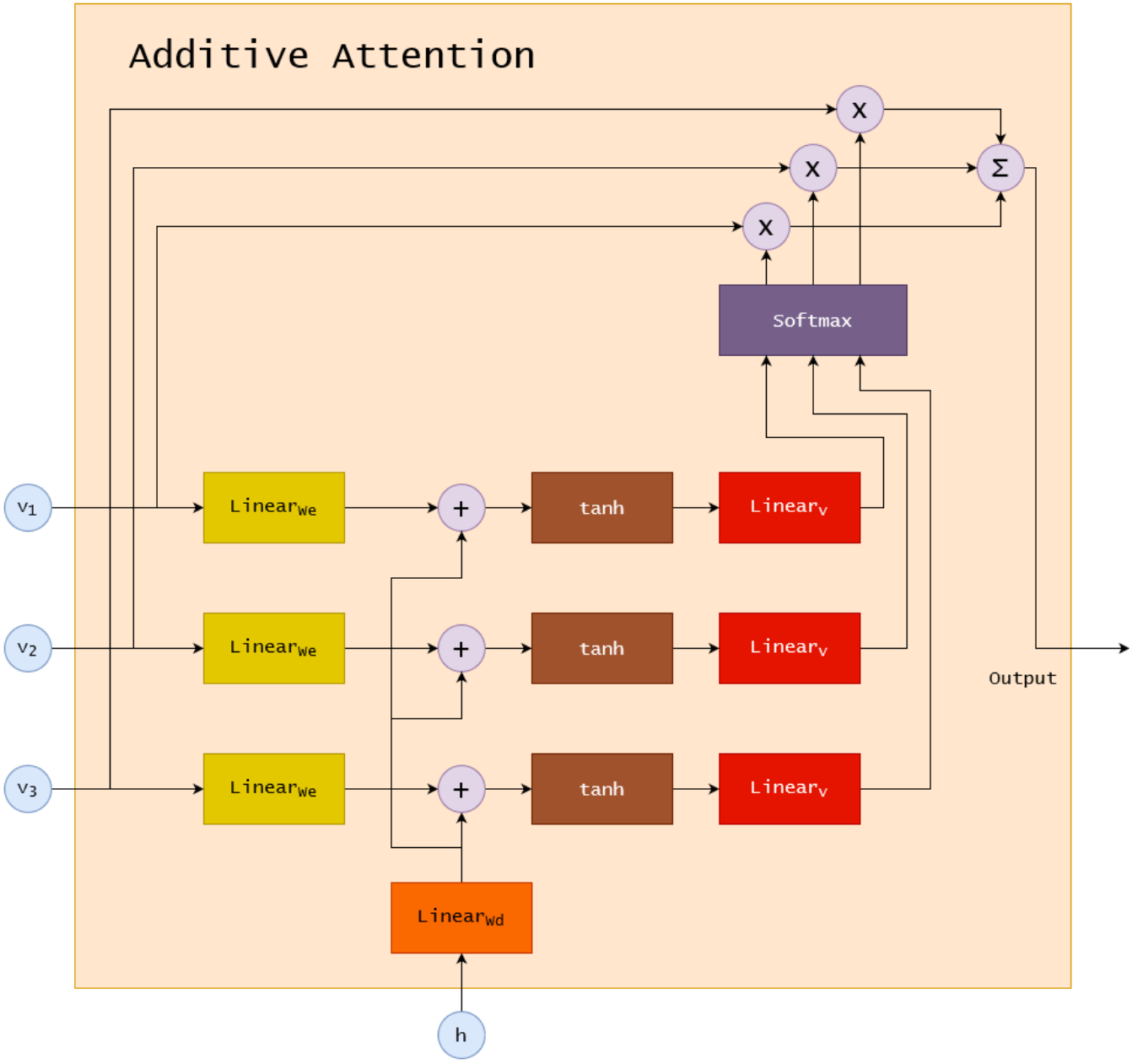


Повечето компоненти ползват готови реализации на Pytorch. Изключение е **Additive Attention** компонента, който е реализиран отделно. По-долу е дадена архитектурата му:

- Като аргументи приема произволен брой вектори **v** с размерност **2 * encoderHiddenSize** и един вектор **h** с размерност **decoderHiddenSize** (размерностите могат да бъдат различни,

това са само стойностите им, който се използват в цялостната архитектура). Резултатът е вектор **output** с размерност **2 * encoderHiddenSize**.

- Линейна трансформация $\textbf{Linear}_{\textbf{we}}$. Тя приема вектор с размерност **2 * encoderHiddenSize** и връща вектор с размерност **attentionHiddenSize**.
- Линейна трансформация $\textbf{Linear}_{\textbf{wd}}$. Тя приема вектор с размерност **decoderHiddenSize** и връща вектор с размерност **attentionHiddenSize**. Освен това няма bias.
- **tanh**, която пресмята хиперболичен тангенс върху елементите на векторите.
- Линейна трансформация $\textbf{Linear}_{\textbf{v}}$. Тя приема вектор с размерност **attentionHiddenSize** и връща вектор с размерност **1**. Освен това няма bias. Това е еквивалентно на скаларно умножение с вектор.
- **Softmax** за нормализиране на вероятностите.
- Умножаване на векторите **v** с получените вероятности и сумиране, което дава изходния вектор.



Използвани стойности на параметрите

Обученият модел използва следните стойности на параметрите:

- **wordEmbeddingSize = 256**
- **encoderHiddenSize = 256**
- **encoderLayers = 4**
- **decoderHiddenSize = 2 * encoderHiddenSize = 512**
- **decoderLayers = encoderLayers = 4**
- **attentionHiddenSize = 256**
- **dropout = 0.3**
- **projectionTransformSize = 1024**

Вижте най-накрая за информация какви експерименти с параметрите са правени и защо са подбрани тези стойности.

Обучение на модела

Проектът използва прикачените помощни програми. Единствените промени по тях са параметрите, които се подават за създаване на модела. Съответно подготовката на данните се извършва с **“python run.py prepare”** командата, а тренирането с **“python run.py train”** командата. Във файла **“parameters.py”** са използвани следните параметри за обучение:

- **learning_rate = 0.001**
- **clip_grad = 5.0**
- **learning_rate_decay = 0.5**
- **batchSize = 32**
- **maxEpochs = 12**
- **log_every = 40**
- **test_every = 2000**
- **max_patience = 3**
- **max_trials = 5**

Самият модел се обучава непрекъснато през всичките 12 епохи. От тях само първите 6 епохи бяха достатъчни, докато се достигнат стойности на перплексията, които са близки до крайната. През останалите 6 епохи имаше сравнително малки оптимизирания в търсене на локален минимум. За обучението съм използвал Google Colab и 12-те епохи сумарно отнеха около 3 часа време.

Оценка на модела

За оценка на модела могат да се използват описаните в заданието операции **“python run.py perplexity <sourceCorpus> <targetCorpus>”**, **“python run.py translate <sourceCorpus> <resultCorpus>”** и **“python run.py bleu <targetCorpus> <resultCorpus>”**.

Освен това в моделът е реализирана възможността за алчно търсене и за търсене по лъча при генериране. Във файла **“parameters.py”** по подразбиране са зададени следните стойности:

- **use_beam_search = True**
- **branching_factor = 4**

Ако **use_beam_search** е **True**, то ще се използва търсене по лъча, за **False** ще се използва алчно търсене. При търсене по лъча може да се избере коефициентът на разклонение **branching_factor**. Силно препоръчително е да се използва коефициент **3** или **4**. Те са достатъчни, за да се получи почти максимална полза от търсенето по лъча. При по-големи стойности има незначително подобрение на BLEU резултата, но времето за изчисление нараства значително.

Крайният модел дава следните резултати:

dev корпус:

- Perplexity –> 4,93
- BLEU с алчно търсене –> 41,27
- BLEU с търсене по лъча, разклонение 4 –> 42.12

test корпус:

- **Perplexity –> 5,39**
- BLEU с алчно търсене –> 39,03
- **BLEU с търсене по лъча, разклонение 4 –> 40,71**

В папката **“results”** са приложени получените преводи от търсенето по лъча с разклонение 4, на които съответстват двата BLEU резултата за dev и test.

Настройване на параметрите на модела

Параметърът **dropout** е най-особен, понеже не е ясно какви стойности са подходящи. За него главно съм направил експерименти с умерена стойност **dropout=0,3** и висока стойност **dropout=0,5** при по-малки стойности за останалите параметри. Като цяло резултатите са близки, но 0,5 винаги дава малко по-лоши резултати върху dev и test множествата. Изглежда високата регуларизация не помага и затова в крайния модел съм избрал умерената стойност 0,3.

За останалите параметри като цяло зависимостта в експериментите е, че по-големи стойности дават по-добри резултати. Избрал съм колкото се може по-големите стойности, за които моделът да се обучава в разумно време. Освен това съм подбрал размерностите да са близки по порядък една до друга, защото интуитивно така се намалява вероятността една от тях да се превърне в “bottleneck” за останалите, а в същото време така никоя от величините не допринася за твърде голям процент от общото време, което е нужно за обучение.

Има особеност с избора на величината **projectionTransformSize**. Компонентът $\mathbf{Linear_T}$ приема вектор с размер **2 * encoderHiddenSize + decoderHiddenSize = 1024** и връща вектор с размер **projectionTransformSize = 1024**. По принцип би било добре размерът на проекцията да е по-голям от този на входния вектор, но с тази промяна моделът започва да става твърде тежък и е нужно значително повече време да се обучи. Съответно изборът на стойност за размер на проекцията е компромис с цел бързодействие.