ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(СПбГУ)

Образовательная программа бакалавриата «Науки о данных»

# Отчёт по практике

**Учебная практика (научно-исследовательская работа)**

Выполнил студент 3 курса бакалавриата
(группа 22.Б05-мкн)
**Михеев Артём Антонович**

Санкт-Петербург

2025

# Построение обратимых эмбеддингов предложений

## Содержание

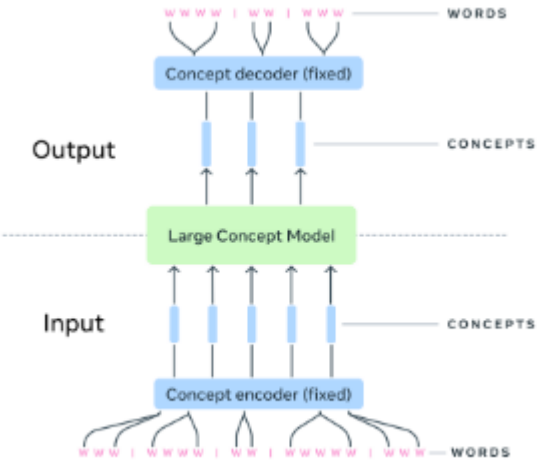
Введение

Обращать эмбеддинги токенов довольно просто. Находимэмбеддинг  в матрице эмбеддингов и номер строчки является порядковым номером токена. Используя токенизатор декодируем и получаем символы.

Предложений бывает очень много, и никакой памяти не хватит для хранения всех эмбедингов. При этом условная обратимость бывает полезной(обратить полностью не получится, так как какая-то часть информации теряется, но получить схожее по смыслу хотелось бы)

Обратимость эмбеддингов предложений полезна в генерации текста, интерпретации, анализе данных и обучение моделей.

Набирает популярность LCM (Большие концептуальные модели), в которых модель учится переводить одни концепты(предложения) в другие. А encoder (из слов в эмбеддинг предложения), decoder(из эмбеддинга предложений в слова) обучаются независимо для каждого языка.



Основные результаты практики

**Подготовка данных.**

Использовался датасет "bentrevett/multi30k” из библиотеки Datasets от Hugging Face. Были оставлены предложения на английском с количеством слов от 5 до 30. (Пример:'a blond girl in a purple shirt and plaid shorts playing with a string toy.')

Предоученные токенизатор и эмбеддинг токенов использовались от моделей:

"prajjwal1/bert-tiny" (emb_size=128 )
"huawei-noah/TinyBERT_General_4L_312D"(emb_size=312)
"sentence-transformers/paraphrase-mpnet-base-v2"(emb_size=768)

## Первая реализация.

Архитектура:
Модель состоит из простых rnn: encoder и decoder. Encoder на вход получает тензор размера BxT, где B-размер батча(=32), T - количество токенов в предложении (с padding токенами).
На выходе(последний выход из rnn) тензор эмбеддингов предложений размера
Bx sentence_emb_size
Decoder на вход для каждого T получает тензор эмбеддингов предложений. На выходе для каждого T линейный слой переводит в эмбединги токенов (BxTx token_emb_size).

Для B=32, T=25, token_emb_size = 128, sentence_emb_size = 512
encoder:[32, 25, 128] -> [32, 512]
decoder+linear: [32,512]->[32,25,512]->[32, 25, 128]

Loss, accuracy:

Для каждого эмбеддинга токена на выходе считается расстояние со всеми эмбеддингами из матрицы эмбеддингов. Если минимальное расстояние с изначальным токеном, то в loss не учитываем.
Иначе mse(input_token_emb, output_token_emb). При этом в силу трудоемкости близость считается как cos_similarity. В качестве ответа берём ближайшей по cos_similarity. Accuracy - доля выходных токенов, ближайшие к которым  входные токены.

```
def match_loss(tokens, word_emb, output_word_emb, all_emb):
  with torch.no_grad():
    output_word_emb_norm = output_word_emb /
output_word_emb.norm(dim=-1, keepdim=True)
    all_emb_norm = all_emb / all_emb.norm(dim=-1, keepdim=True)
    best_similarity =  torch.matmul(output_word_emb_norm,
all_emb_norm.T).argmax(-1)
    mask =  best_similarity != tokens

  loss = (mask * ((output_word_emb-word_emb)**2).sum(-1)).sum()
  return loss, (best_similarity == tokens).float().sum()
```

```
EPOCH 0
Train_loss=0.7488839063390808, Val_loss=0.547196175785717
Train_accuracy=0.15504117452314975, Val_accuracy=0.04486268053580783
EPOCH 1
Train_loss=0.3687221738381444, Val_loss=0.4391373728230871
Train_accuracy=0.14464250683337038, Val_accuracy=0.054809505846850244
```

Само расстояние между выходом и входом уменьшается, однако ближайший из emb_matrix редко является input_emb, так как из меньшего mse, не следует больший cos_sim. (Upd:В статье [1] решили проблему.)

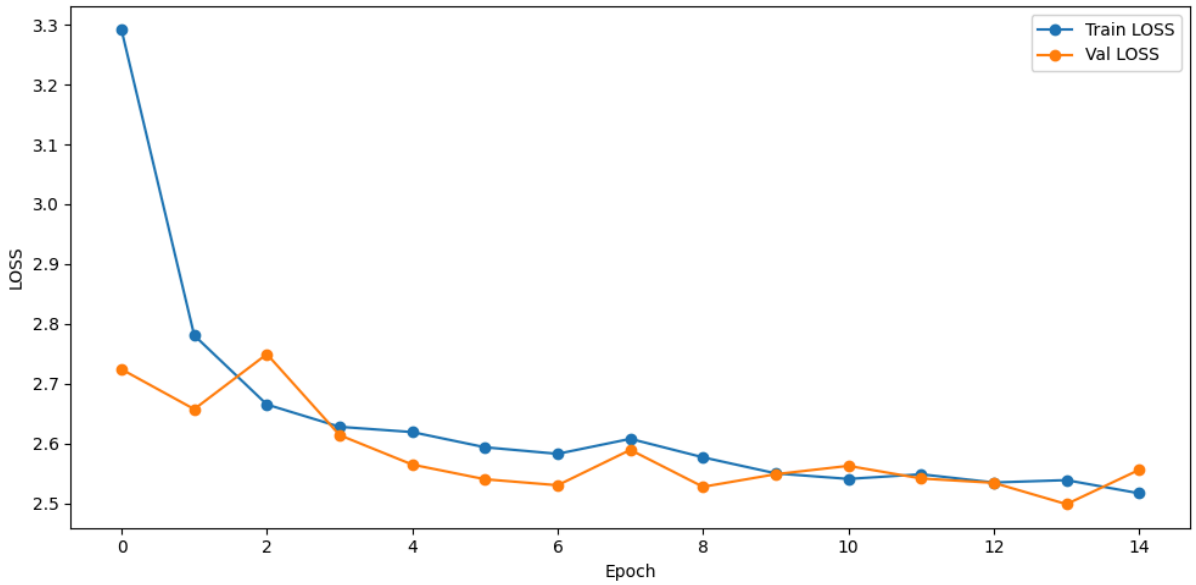
**Вторая реализация.**

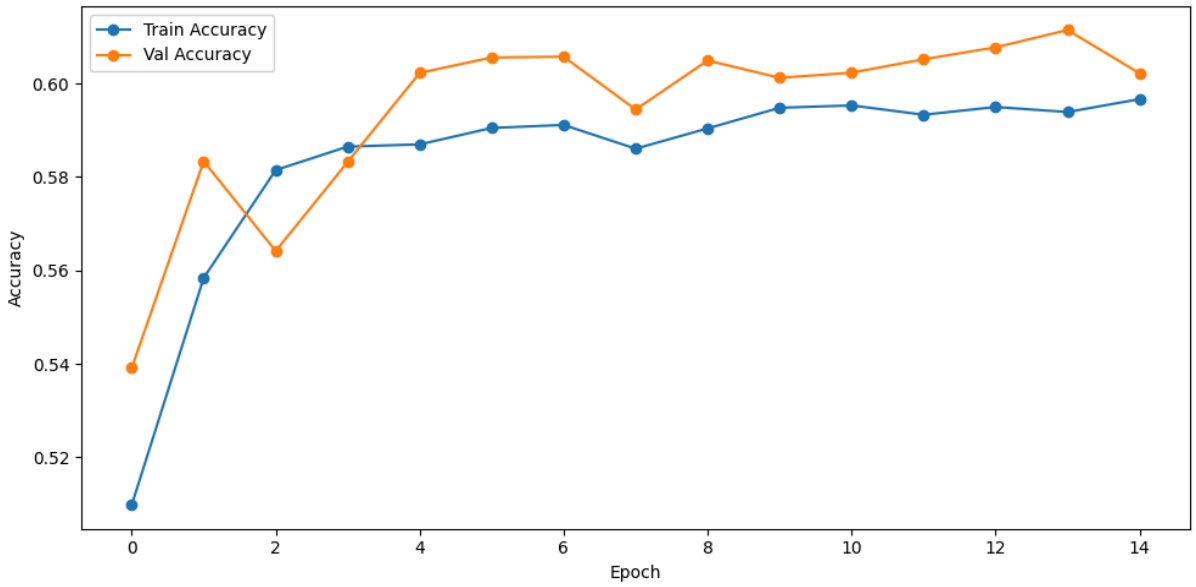
В следующей реализации используем encoder такой же, а decoder на выходе после применения линейного слоя получает logits на vocab_size токенов. (output_size = BxTx vocab_size)

```
loss = F.cross_entropy(
            logits.reshape(-1, len(tokenizer)),
            tokens.reshape(-1),
        )
```

Сливаем размерность батча и длины предложения в одну для подачи в F.cross_entropy

После обучения 15 эпох с Adam_optimizer(learning rate = 1e-4, weight_decay=1e-5), batch_size = 32, с предобуенными эмбеддингами токенов размера 312 из "huawei-noah/TinyBERT_General_4L_312D" и размера эмбеддинга предложений 1024 получилось так:




Теперь можно декодировать генерировать токены из logits как мультиномиальное распределение.

**Дополнения**:

Использование усложнений rnn(bidirectional lstm rnn)

Можно добавить loss не на уровне токенов, а на уровне предложений. Так как мы хотим получить предложение похожее по смыслу можно считать cosine_similarity как их схожесть и оптимизировать к 1.

Как генерировать из logits эмбеддинги токенов, чтобы градиент шёл. Сэмплирование само по себе не пропускает градиент, поэтому нужно воспользоваться трюком с репараметризацией.

Gumbel-softmax trick:

X~[p1,p2,..,pk] = argmax(gi+logpi), gi~Gumbel

Генерировать из распределения Гумбеля можно так(Функция распределения F(x) имеет распределение U(0,1) и берем обратную функцию):

```
p = (-torch.log(-torch.log(torch.rand(probs.shape,device=device) + 1e-20) + 1e-20))
```

Вместо argmax будем брать его гладкую версию soft_max и для получения эмбеддингов токенов сложим все эмбеддинги с весами. T -температура softmax, при

T < 1 максимум явно выражен, T>1 равномернее.

```
probs = F.softmax(logits, dim=-1)

alpha = F.softmax((p+torch.log(probs))/T, dim=-1)
```

1)Теперь можно сгенерированные эмбеддинги токенов вновь подать на вход encoder, получить новый эмбеддинг предложения и считать cos_similarity  между ним и выходом encoder для начальных токенов.

2)Использовать готовую модель, строящую эмбеддинги предложений с cos_similarity около 1 для похожих по смыслу предложений. Такой моделью является "paraphrase-mpnet-base-v2" из библиотеки sentence_transformers.

///(Она обучалась на задачах Semantic Textual Similarity, Paraphrase Identification.

MPNet обучается на больших корпусах текстов с использованием задач MLM.

Дообучение включает:Triplet Loss или Cosine Similarity Loss:

Триплетная функция потерь использует три предложения: анкер, положительный пример и отрицательный пример. Цель — минимизировать расстояние между анкером и положительным примером и максимизировать его для отрицательного.

Косинусное сходство измеряет, насколько похожи два вектора, и используется для регрессии (например, для STS).///

В этом случае, если хотим использовать свои эмбеддинги токенов можно обучить модель, получающую из них эмбеддинг предложения из sentence_transformers.

Второй вариант использовать эмбеддинги из "paraphrase-mpnet-base-v2" и так как она на вход принимает токены,  изменяем вход слоя после слоя эмбеддингов.

```
EPOCH 0

Train_loss=3.9825067124612827, Val_loss=3.5035101814313196

Train_accuracy=0.5269569252265309, Val_accuracy=0.5848975848364415
```

Модель обучается быстрее, но трудно затратнее. В итоге accuracy около 0.61

# Заключение

Были изучены архитектуры на основе рекуррентных сетей и реализованы модели для построения обратимых эмбеддингов предложений. Дальнейшие исследования могут быть направлены на улучшение архитектур моделей и применение новых методов оптимизации для повышения качества обратимых эмбеддингов. Обучение своих моделей типа paraphrase-mpnet-base-v2 для лучшего loss. Использование данных из конкретной области (финансовые новости). Изучение пространства эмбеддингов SONAR для LCM из статьи [3] , вышедшей в конце декабря.

# Список использованных литературных источников и информационных материалов


[1] Residual Recurrent Networks for Invertible Sentence Embeddings
[2] Paraphrase Dataset Infused with High-Quality Lexical and Syntactic Diversity
[3] Large Concept Models