Машинен превод чрез генеративен езиков модел

Име: Мария Гроздева

Факултетен номер: 6MI3400675

1. Архитектура на модела

Реализацията е базирана на decoder-only Transformer.

Основните компоненти в model.py са:

- **Embedding слой** – map-ва токени към вектори с размер d_model.
- **Positional Encoding** – синус/косинус позиционно кодиране (както в оригиналния Transformer), добавя се към embedding-ите.
- **Многоглаво self-attention (Multi-Head Attention)** – използвам n_heads глави, като проекциите са Wq, Wk, Wv и Wo.
- **Causal mask + pad mask** – за да не „вижда“ бъдещи токени при генериране, и да игнорира PAD токените.
- **Position-wise feed-forward** – две линейни проекции с ReLU: d_model → d_ff → d_model.
- **Dropout + LayerNorm** – след attention и след FFN, със residual връзки за по-успешно трениране.

Самият Decoder е стек от n_layers еднакви DecoderLayer блока. След последния слой има линейна проекция към речника (vocab), която дава logits за следващия токен.

Загубата е cross-entropy върху следващия токен (teacher forcing), като PAD токенът се игнорира.

Параметри (всички са в parameters.py):

- n_layers (брой decoder layers): 6
- n_heads (брой attention глави): 16
- d_model (размер на embedding): 512
- d_keys, d_values (размери на ключове/стойности на глава): d_model // n_heads

- d_ff (размер на feed-forward слоя): 2048 (taken from the original paper)
- seq_len (макс. дължина на последователността): 400
- dropout: 0.1
- learning_rate: 0.0003
- batchSize: 32
- maxEpochs: 10
- clip_grad: 5

2. Обучение и експерименти

Данните се подготвят с run.py prepare: прочитам source/target корпусите, tokenization е с NLTK word_tokenize. След това изграждам речник (word2ind) от source+target, като редките думи (под праг 3) се мапват към <UNK>. Тренировъчните примери са конкатенация: <START> + source + <TRANS> + target + <END>.

Обучението е с Adam оптимизатор. Loss е cross-entropy върху следващ токен, със ignore_index за PAD.

Експерименти:

• Промяна на dropout и learning rate.
• Различни n_layers/n_heads (повече слоеве дава повече капацитет, но учи по-бавно и иска повече данни).

3. Оценка на тестов корпус: перплексия и BLEU

Оценяването става с командите:
• python run.py perplexity source_test target_test
• python run.py bleu reference_en translation_en
където translation_en е файлът, генериран от python run.py translate.

Резултати:

- **Perplexity:** 3.3769701683274103
- **Corpus BLEU:** 37.593299276232514

4. Източници

- Attention Is All You Need
- Neural Machine Translation by Jointly Learning to Align and Translate
- Massive Exploration of Neural Machine Translation Architectures
- Optimizing Transformer for Low-Resource Neural Machine Translation
- PyTorch
- Transformers tutorials