

Многозадачное обучение языковых моделей, основанных на механизме внутреннего внимания

Погорельцев С. А., н.р. Полякова И. Н.

МГУ им. Ломоносова, ф-т ВМК, каф. АЯ

2022

Многозадачное обучение

Многозадачное обучение (MultiTask Learning) - подполе машинного обучения, в котором одновременно решаются несколько задач, при этом используются общие черты и различия между задачами. Это может привести к повышению эффективности обучения и точности прогнозирования для моделей, специфичных для конкретных задач, по сравнению с отдельным обучением моделей.

Языковые модели, основанные на механизме внутреннего внимания

Да да, я про Трансформеры и в частности BERT-подобные модели

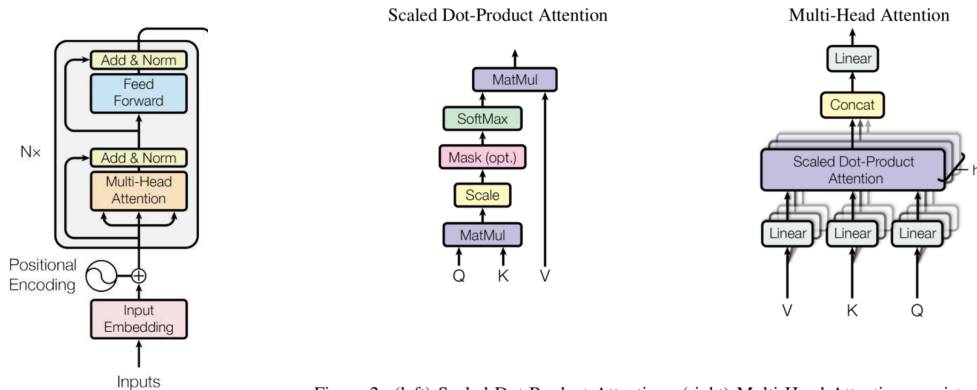


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

Формализация понятия задача

$$T_i = \{p_i(x), p_i(y|x), L_i\} \quad (1)$$

- $p_i(x)$ - распределение входных данных
- $p_i(y|x)$ - распределение меток (классификация) / значений (регрессии) в зависимости от входных данных
- L_i - функция потерь

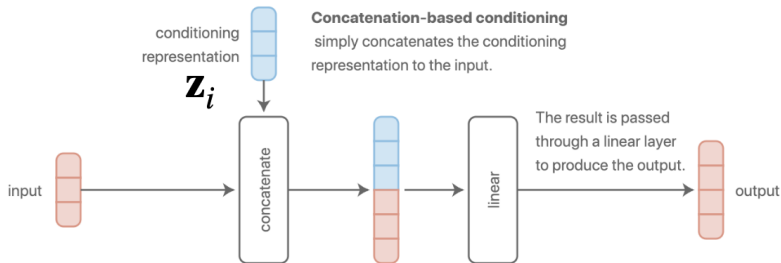
Выборки для задачи

- D_i^{train} - обучающая
- D_i^{val} - валидационная
- D_i^{test} - тестовая

Гипотезы многозадачной модели (Hard Parameter Sharing)

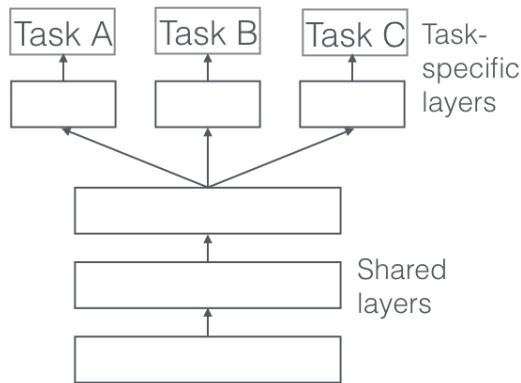
Основной вопрос: Как организовать "переключение" на уровне модели?

- Конкатенация one-hot вектора задачи к скрытому представлению



- Добавление вектора смещения - смещение данных задачи
- Умножение - изменение масштаба данных задачи
- Архитектура с несколькими головами (популярна, хорошо работает на практике, в дальнейшем буду говорить о ней, т. к. использую её в работе)

Архитектура с несколькими головами



Введём некоторые обозначения и уточним понятие задачи

- H_i - специфичные для задачи слои - голова модели
- E - общие слои, формирующие представления для голов H_i

Тогда i -ая задача:

$$T_i = \{p_i(x), p_i(y|x), H_i, L_i\} \quad (2)$$

Forward pass для задачи T_i

$$F_i(x) = H_i \circ E \circ x = H_i(E(x)) \quad (3)$$

Обучение многоголовой модели (популярные подходы)

- **Одна задача на батч** Данные всех задач разбиваются на батчи и эти батчи перемешиваются вместе. На каждой эпохе выбирается случайный j -ый батч соответствующий задаче T_i $batch_j^i$, а затем всё (почти) как обычно.
- **Оптимизация (взвешенной) суммы функций потерь**
 - Собираются "метабатчи" $\{batch_{j_1}^1, batch_{j_2}^2, \dots, batch_{j_n}^n\}$ - множество по 1 случайному батчу для каждой задачи (здесь n штук).
 - Для каждой i -ой задачи вычисляется $fwd_i = F_i(batch_{j_i}^i)$, $loss_i = L_i(f_i)$ и оптимизируется $loss = \sum_{i=1}^n w_i loss_i$, где $\forall i w_i > 0$.
 - w_i регулирует важность i -ой задачи при оптимизации

Я экспериментирую с обоими и кажется, что оптимизация суммы работает лучше
~~что в целом логично, т. к. первый вариант с точки зрения метоптов – тихий ужас~~

Алгоритм обучения с оптимизацией по одной задаче

```
for epochNum  $\leftarrow \overline{0, N}$  do
  for all  $batch_j^i \in batches.shuffle()$  do
     $outputs \leftarrow F_i(batch_j^i)$ 
     $loss \leftarrow L_i(outputs)$ 
     $F_i \leftarrow F_i - \alpha \nabla loss$ 
  end for
end for
```

▷ Тут может быть любой метод оптимизации

Minibatch градиентный спуск здесь для упрощения примера, чаще используется Adam или AdamW для обучения Трансформеров (в т.ч. многозадачных)

Алгоритм обучения с оптимизацией по сумме функций потерь

```
for epochNum  $\leftarrow \overline{0, N}$  do ▷ Итерация по эпохам  
  for all metabatchj  $\in$  metabatches.shuffle() do ▷ Итерация по "метабатчам"  
    loss  $\leftarrow 0$   
    for all batchji  $\in$  metabatchj do ▷ Итерация по задачам и их батчам  
      outputsji  $\leftarrow F_i(\text{batch}_j^i)$   
      lossj  $\leftarrow \text{loss}_j + w_i L_i(\text{outputs}_j^i)$   
    end for  
    lossj.backward(); optimizer.step(); optimizer.zero_grad() ▷ *  
  end for  
end for
```

*: магия автоматического дифференцирования PyTorch, а на самом деле

$$\nabla \text{loss}_j(x^{i,j}) = \nabla \sum_{i=1}^n w_i L_i(H_i(E(x^{i,j}))) \text{ (продолжение на след. слайде)}$$

Подробнее про обновление весов для суммы

Рассмотрим частную производную для k -ой координате

$$(\nabla loss_j(x^{i,j}))_k = \frac{\partial}{\partial x_k^{i,j}} \sum_{i=1}^n w_i L_i(H_i(E(x^{i,j}))) = \sum_{i=1}^n w_i \frac{\partial L_i(H_i(E(x^{i,j})))}{\partial x_k^{i,j}}$$

Дальше считается как обычно, как и в предыдущем случае градиент от конкретной функции потерь с использованием правила цепочки $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z} \frac{\partial z}{\partial x}$

После вычисления градиента делается шаг метода оптимизации обновляются все веса и для общих слоёв, и для **каждой головы** (backpropagation)

- Полученный метод - обобщение первого (если w - one-hot вектор, то получаем первый метод буквально)
- Снижается проблема забывания, поскольку оптимизация всего сразу
- Возможно, стоит совместить первый и второй подходы
 - Сначала сделать базовую оптимизацию для всех задач
 - Затем заниматься более тонкой настройкой позадачных весов

Некоторые важные особенности

- **Transfer Learning:** знания для одной задачи помогают в решении другой
- **Representation Learning:** перенос знаний часто связан с выучиванием общими слоями хороших векторных представлений входных признаков
- **Регуляризация:** многозадачное обучение работает как регуляризация и в случае переобучения всегда можно добавить задач для улучшения генерализации или сделать общими больше слоёв
- **Эффективное использование:** модель одна, пусть и крупнее

Сложности

- **Negative Transfer:** бывает, что задачи отрицательно влияют на решение друг друга. Наиболее вероятные причины:
 - Недостаточно выразительная модель (часто многозадачные модели больше)
 - Сложности с оптимизацией (межзадачная интерференция, задачи могут обучаться с разной скоростью)
- **Совместность задач:** какие задачи будут хорошо работать вместе? На этот вопрос ответ пока можно искать только экспериментами (за 1 эпоху понятно)

Проблема со средствами многозадачного обучения

Во время первых экспериментов обнаружил, что приходится писать много однообразного и сложного кода, потому, что хороших библиотек для многозадачного обучения Трансформеров пока нет (или я плохо искал). В итоге, была разработана (и продолжает развиваться) небольшая библиотека, которая позволяет декларативно описывать задачи и сама собирает модель, готовит данные с учётом многозадачности

- Высокоуровневые готовые конфигурации для типичных задач
- "Ручки" для низкоуровневой настройки задач
- Средства для обучения в обычном цикле PyTorch
- Обучение с многозадачным аналогом Trainer (сейчас перерабатывается)

Пример простой конфигурации задач

```
tasks = Tasks([
    SequenceClassificationTask(
        name="danetqa",
        dataset_dict=load_dataset(...),
        preprocessor=Preprocessor([preprocess_danetqa]),
        tokenizer_config=TokenizerConfig(max_length=512),
    ),
    SequenceClassificationTask(
        name="headline_cause",
        num_labels=3,
        dataset_dict=load_dataset(...),
        preprocessor=Preprocessor([preprocess_headline_cause]),
        tokenizer_config=cfg
    )], model_path = "DeepPavlov/rubert-base-cased")
```

Низкоуровневая конфигурация задачи

```
Task(  
    name = "название задачи (для индексации и логов)",  
    head = <объект подкласса torch.nn.Module> | HFHead,  
    data = Data(  
        dataset_dict = <датасет в формате datasets.DatasetDict>,  
        # другое (не обязательно, задано по-умолчанию)  
        configured_tokenizer = ConfiguredTokenizer(  
            model_path, padding = False, truncation = True, max_length  
            ... # другие параметры токенизатора Hugging Face  
        ),  
        preprocessor = Preprocessor(...),  
        columns, # поля, которые нужно приводить к формату PyTorch  
        collator_class, collator_config  
    )  
)
```

Конфигурация головы модели

- Готовая из transformers

```
head = HFHead(class_ = AutoModelForSequenceClassification,  
              config_params = {"num_labels": 2})
```

- Свой torch.nn.Module

```
class TwoLinears(nn.Module):  
    def __init__(self, ...):  
        layers = [nn.Linear(768, 2048), F.relu, nn.Linear(768, 2)]  
        self.layers = nn.ModuleList(layers)  
        self.loss = nn.CrossEntropyLoss()  
    def forward(self, encoder_outputs, labels, ...):  
        x = encoder_outputs[1]  
        for layer in self.layers: x = layer.forward(x)  
        return {"logits": x,  
                "loss": self.loss(logits.view(-1, num_labels),  
                                   labels.view(-1))}  
  
head = TwoLinears(...)
```

Обучение с циклом на PyTorch

```
train_sampler = MultitaskBatchSampler(tasks.data, "train", batch_size=12)
model = MultitaskModel(encoder_path, tasks.heads)
# ... инициализация метода оптимизации, перенос модели на GPU ...
for epoch_num in range(num_epochs):
    for batch in train_sampler:
        batch.data.to(device)
        outputs = model.forward(batch.name, **batch.data)
        loss = outputs.loss
        print(f"Training loss: {loss} on task {batch.name}")
        loss.backward()
        optimizer.step()
        lr_scheduler.step()
        optimizer.zero_grad()
```


Готовность и открытый доступ

- Код в открытом доступе
<https://github.com/s1m0000n/multitask-transformers>
- Буду рад новым контрибьюторам
- Нормальное состояние кодовой базы (pylint > 8)
- Документация в зачаточном состоянии
- Новые вещи реализуются по мере того, что я использую для ВКР
- Планируется публикация статьи

Цели и планы (неформально)

- Попытаться исправить / уменьшить проблемы оптимизации (важно)
- Решение проблемы Negative Transfer усложнением голов (важно)
- MTL как средство улучшения качества на малых выборках (важно)
 - В первоначальных экспериментах удалось улучшить качество задачи DaNetQA
 - Обучалось вместе с задачей NLI с позадачной оптимизацией
 - Планируется рассмотреть подробнее, после прочих улучшений
- Рассмотрение многозадачных трансформеров для русского (как получится)
- Попробовать свои силы в соревновании [Russian SuperGLUE](#) (если успею)

Сходимость и достижимость результатов однозадачных моделей

Сходимость даже с деревянной оптимизацией

Резы достижимы

Графичек

Особенности способов оптимизации

- Обучение с оптимизацией по взвешенной сумме лоссов
 - Стабильная сходимость
 - Грубо оптимизирует всё сразу
 - Тяжело достичь очень высоких результатов на задачах (согласно метрикам)
 - Гипотеза: хорошо подойдёт для "грубой" и быстрой начальной оптимизации
- Обучение с позадачной оптимизацией
 - Медленная и нестабильная сходимость
 - Наблюдается следующее поведение
 - Хорошо пооптимизировал некоторую задачу
 - Одновременно в разной степени просел на других
 - Получаем в целом медленное скачкообразное улучшение лоссов и метрик
 - Гипотеза: хорошо подойдёт для "тонкой" донастройки по задачам, при условии, что проведена начальная "грубая" оптимизация

Предлагаемое улучшение подхода

- ❶ Делаем первоначальную оптимизацию: пока не выйдем на плато - обучаемся по взвешенной сумме функций потерь
- ❷ Далее - тонкая оптимизация, варианты
 - Просто позадачная оптимизация (выше вероятность проблемы забывания)
 - Цикл: N раз выполняем позадачную оптимизацию, а затем один раз по взвешенной сумме (борьба с забыванием)

Так же при первоначальной и точной оптимизациях можно иногда пропускать обучение на задачах, которые по лоссу / метрикам уже оптимизировались значительно лучше, чем остальные

Это может помочь побороть проблему с разной скоростью сходимости задач

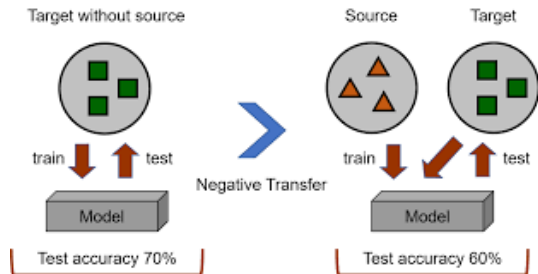
Это пока просто гипотеза, но я её обязательно проверю

Результаты первых экспериментов

- Рассматривается фиксированный набор задач (TODO)
- Такие то такие то параметры (TODO)

Результаты: Табличка (TODO)

Подробнее о проблеме Negative Transfer

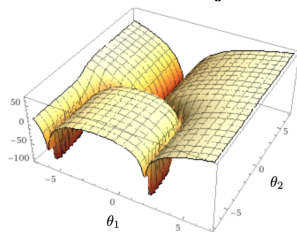


Основные причины

- Проблемы оптимизации
 - Задачи могут учиться с разной скоростью
 - Получается слишком сильная регуляризация (регулируем гиперпараметры)
 - Большой угол между градиентами
- Недостаточная выразительность модели - обычно многозадачные модели значительно крупнее, чем однозадачные и требуется усложнять гипотезы голов

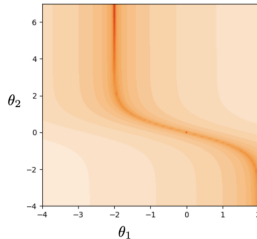
Проблема отличающихся целей

Multi-Task Objective



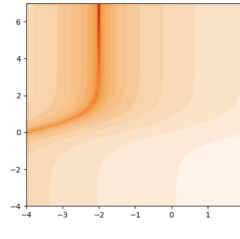
(a)

Task 1 Objective



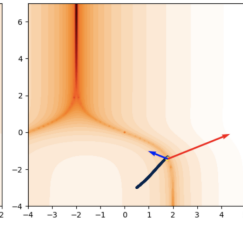
(b)

Task 2 Objective



(c)

Adam



(d)

Yu et al. Gradient Surgery for Multi-Task Learning. 2020

Решение

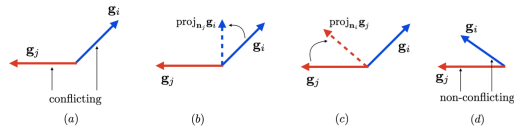
Algorithm 1 PCGrad Update Rule**Require:** Model parameters θ , task minibatch $\mathcal{B} =$ $\{\mathcal{T}_k\}$ 1: $\mathbf{g}_k \leftarrow \nabla_{\theta} \mathcal{L}_k(\theta) \quad \forall k$ 2: $\mathbf{g}_k^{\text{PC}} \leftarrow \mathbf{g}_k \quad \forall k$ 3: **for** $\mathcal{T}_i \in \mathcal{B}$ **do**4: **for** $\mathcal{T}_j \stackrel{\text{uniformly}}{\sim} \mathcal{B} \setminus \mathcal{T}_i$ in random order **do**5: **if** $\mathbf{g}_i^{\text{PC}} \cdot \mathbf{g}_j < 0$ **then**6: *// Subtract the projection of \mathbf{g}_i^{PC} onto \mathbf{g}_j* 7: Set $\mathbf{g}_i^{\text{PC}} = \mathbf{g}_i^{\text{PC}} - \frac{\mathbf{g}_i^{\text{PC}} \cdot \mathbf{g}_j}{\|\mathbf{g}_j\|^2} \mathbf{g}_j$ 8: **return** update $\Delta\theta = \mathbf{g}^{\text{PC}} = \sum_i \mathbf{g}_i^{\text{PC}}$ 

Figure 2: Conflicting gradients and PCGrad. In (a), tasks i and j have conflicting gradient directions, which can lead to destructive interference. In (b) and (c), we illustrate the PCGrad algorithm in the case where gradients are conflicting. PCGrad projects task i 's gradient onto the normal vector of task j 's gradient, and vice versa. Non-conflicting task gradients (d) are not altered under PCGrad, allowing for constructive interaction.

Гипотеза 1: стандартная

- Рассмотрим задачу i -ую классификации последовательности $T_i = \{p_i(x), p_i(y|x), H_i, L_i\}$, где $p_i(x), p_i(y|x)$ - распред. входов, меток
- Функция потерь $L_i(p, y) = -\sum_c^{C_i} \delta_{c,y} \log p_c$ (кросс-энтропия)
- Зафиксируем гипотезу для общих слоёв E - стек энкодеров Трансформера
- $\dim E$ - размерность выходных векторов; $C_i = 1, 2, \dots, K_i, K_i$ - число классов
- Прямой проход по нейронной сети для задачи i : $F_i(\text{seq}) = H_i \circ E_{\text{CLS}} \circ \text{seq}$

Гипотеза

$$H_i^1 = \text{softmax} \circ \text{linear}_i$$

- $\text{linear}_i(x) = W_i[1; x]$; $\text{linear}_i : R^{\dim E} \rightarrow R^{K_i}$
- $\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_{k=1}^{K_i} \exp(x_k)}$; $\text{softmax} : R^M \rightarrow R^M$

Гипотеза 2: стек линейных слоёв

- Рассмотрим задачу i -ую классификации последовательности
 $T_i = \{p_i(x), p_i(y|x), H_i, L_i\}$, где $p_i(x), p_i(y|x)$ - распред. входов, меток
- Функция потерь $L_i(p, y) = -\sum_c^C \delta_{c,y} \log p_c$ (кросс-энтропия)
- Зафиксируем гипотезу для общих слоёв E - стек энкодеров Трансформера
- $\dim E$ - размерность выходных векторов; $C_i = 1, 2, \dots, K_i, K_i$ - число классов
- Прямой проход по нейронной сети для задачи i : $F_i(\text{seq}) = H_i \circ E_{\text{CLS}} \circ \text{seq}$

Гипотеза

$$H_i^2 = \text{softmax} \circ \text{linear}_{i,L} \circ \dots \circ \text{relu} \circ \text{linear}_{i,2} \circ \text{relu} \circ \text{linear}_{i,1}$$

- $\text{linear}_{i,j}(x) = W_{i,j}[1; x]$
- $\text{linear}_{i,1} : R^{\dim E} \rightarrow R^d$
- $\text{linear}_{i,l} : R^d \rightarrow R^d \quad \forall l = \overline{2, (L-1)}$
- $\text{linear}_{i,L} : R^d \rightarrow R^{K_i}$
- $\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_{k=1}^{K_i} \exp(x_k)}$; $\text{softmax} : R^M \rightarrow R^M$

Гипотеза 3: накинём вниманий

- Рассмотрим задачу i -ую классификации последовательности
 $T_i = \{p_i(x), p_i(y|x), H_i, L_i\}$, где $p_i(x), p_i(y|x)$ - распред. входов, меток
- Функция потерь $L_i(p, y) = -\sum_c^{C_i} \delta_{c,y} \log p_c$ (кросс-энтропия)
- Зафиксируем гипотезу для общих слоёв E - стек энкодеров Трансформера
- $\dim E$ - размерность выходных векторов; $C_i = 1, 2, \dots, K_i, K_i$ - число классов
- Прямой проход по нейронной сети для задачи i : $F_i(\text{seq}) = H_i \circ E_{\text{CLS}} \circ \text{seq}$

Гипотеза

$$H_i^3 = H_i^h \circ \text{subEncoder}_i$$

- $\text{subEncoder}_i = \text{MHSA}_{i,M} \circ \dots \circ \text{MHSA}_{i,2} \circ \text{MHSA}_{i,1}$
- $\text{MHSA}_{i,m} = \text{concat}(\text{head}_1^{(i)}(X), \dots, \text{head}_h^{(i)}(X)) W_O^{(i)}$
- $\text{head}^{(i)}(X) = \text{softmax}\left(\frac{Q(X)K(X)^T}{\sqrt{d_h}}\right) V(X)$
- $Q(X) = XW_Q^{(i)}, K(X) = XW_K^{(i)}, V(X) = XW_V^{(i)}$

Результаты первых экспериментов с разными гипотезами

- Рассматривается фиксированный набор задач (TODO)
- Такие то такие то параметры (TODO)

Результаты: Табличка (TODO)

Текущая таблица лидеров

Rank	Name	Team	Link	Score	LIDIRus	RCB	PARus	MuSeRC	TERRa	RUSSE	RWSD	DaNetQA	RuCoS
1	HUMAN BENCHMARK	AGI NLP	i	0.811	0.626	0.68 / 0.702	0.982	0.806 / 0.42	0.92	0.805	0.84	0.915	0.93 / 0.89
2	Golden Transformer v2.0	Avengers Ensemble	i	0.755	0.515	0.384 / 0.534	0.906	0.936 / 0.804	0.877	0.687	0.643	0.911	0.92 / 0.924
3	YaLM p-tune (3.3B frozen + 40k trainable params)	Yandex	i	0.711	0.364	0.357 / 0.479	0.834	0.892 / 0.707	0.841	0.71	0.669	0.85	0.92 / 0.916
4	ruT5-large finetune	SberDevices	i	0.686	0.32	0.45 / 0.532	0.764	0.855 / 0.608	0.775	0.773	0.669	0.79	0.86 / 0.859
5	ruRoberta-large finetune	SberDevices	i	0.684	0.343	0.357 / 0.518	0.722	0.861 / 0.63	0.801	0.748	0.669	0.82	0.87 / 0.867
6	Golden Transformer v1.0	Avengers Ensemble	i	0.679	0.0	0.406 / 0.546	0.908	0.941 / 0.819	0.871	0.587	0.545	0.917	0.92 / 0.924
7	ruT5-base finetune	Sberdevices	i	0.635	0.267	0.423 / 0.461	0.636	0.808 / 0.475	0.736	0.707	0.669	0.769	0.85 / 0.847
8	ruBert-large finetune	SberDevices	i	0.62	0.235	0.356 / 0.5	0.656	0.778 / 0.436	0.704	0.707	0.669	0.773	0.81 / 0.805
9	ruBert-base finetune	SberDevices	i	0.578	0.224	0.333 / 0.509	0.476	0.742 / 0.399	0.703	0.706	0.669	0.712	0.74 / 0.716
10	YaLM 1.0B few-shot	Yandex	i	0.577	0.124	0.408 / 0.447	0.766	0.673 / 0.364	0.605	0.587	0.669	0.637	0.86 / 0.859
11	RuGPT3XL few-shot	SberDevices	i	0.535	0.096	0.302 / 0.418	0.676	0.74 / 0.546	0.573	0.565	0.649	0.59	0.67 / 0.665
12	RuBERT plain	DeepPavlov	i	0.521	0.191	0.367 / 0.463	0.574	0.711 / 0.324	0.642	0.726	0.669	0.639	0.32 / 0.314

Материалы / источники

- Стэнфордский курс CS 330: Deep Multi-Task and Meta Learning
- An Overview of Multi-Task Learning in Deep Neural Networks
- Yu et al. Gradient Surgery for Multi-Task Learning. 2020
- A Survey on Negative Transfer. 2021