БДЗ2

Add a description...

Konstantin

Изначально выбрали трансформер с такими параметрами:

```
attn_heads:4
dec_layers:3
dim_feedforward:128
dropout:0.1
embed_size:128
enc_layers:3
```

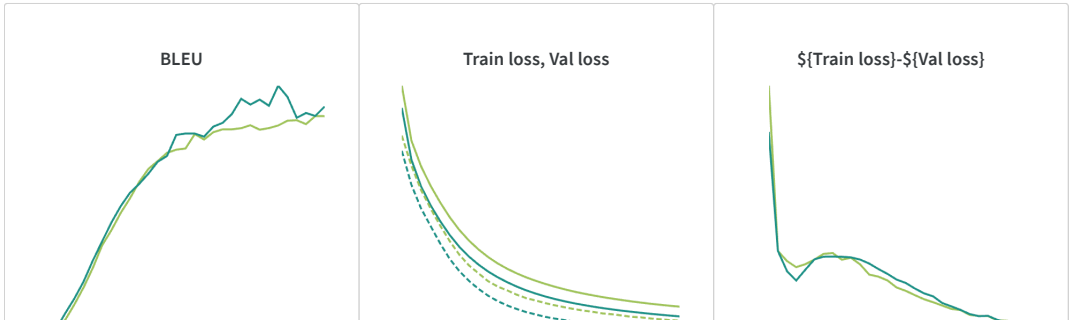
Использовался Adam без расписания:

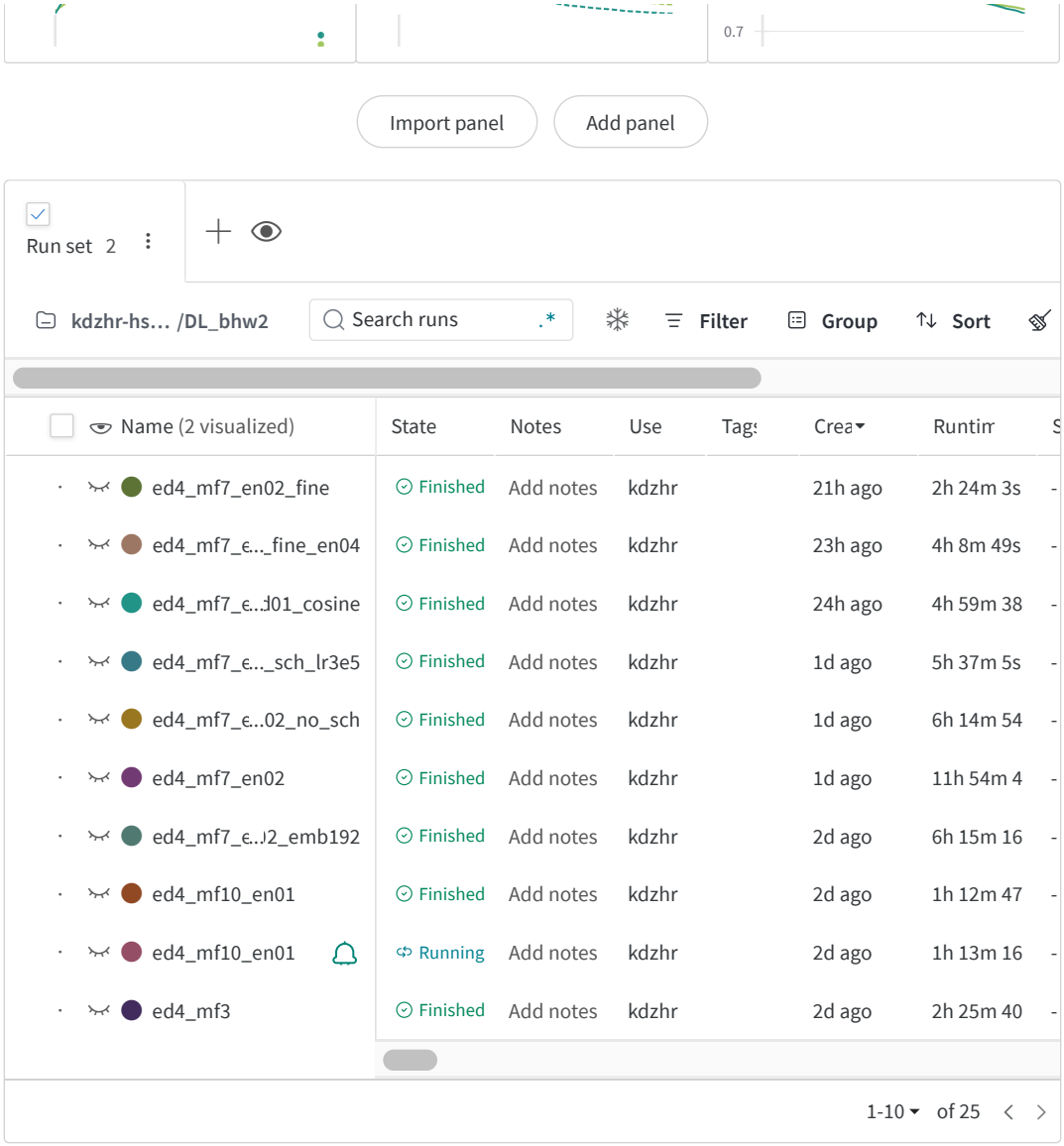
```
batch_size:128
epochs:30
lr:0.0003
src_min_freq:20
tgt_min_freq:20
```

Python

Эксперимент 1

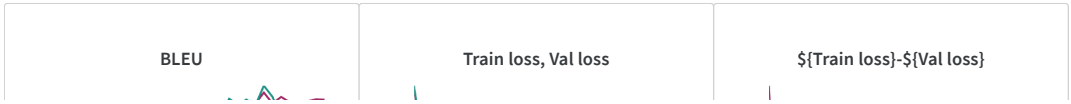
Попробуем увеличить размеры словарей, уменьшив фильтр по min_freq до 5. Видим, что на больших словарях модель учится медленнее, и за 30 эпох лучшего качества не достигает. Модели учатся долго, нужно что-то делать с lr и расписанием.

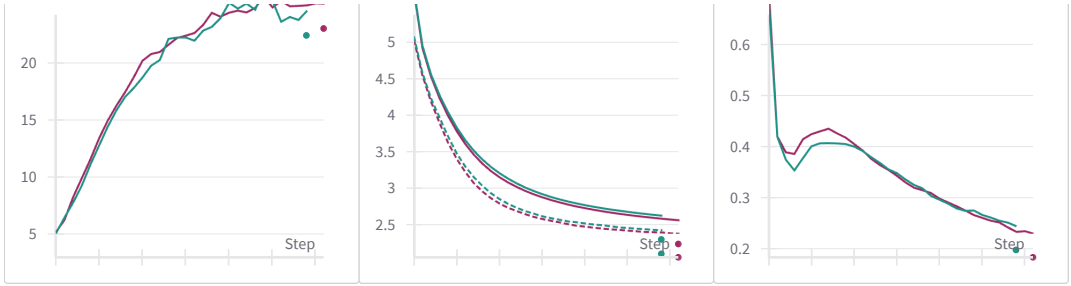




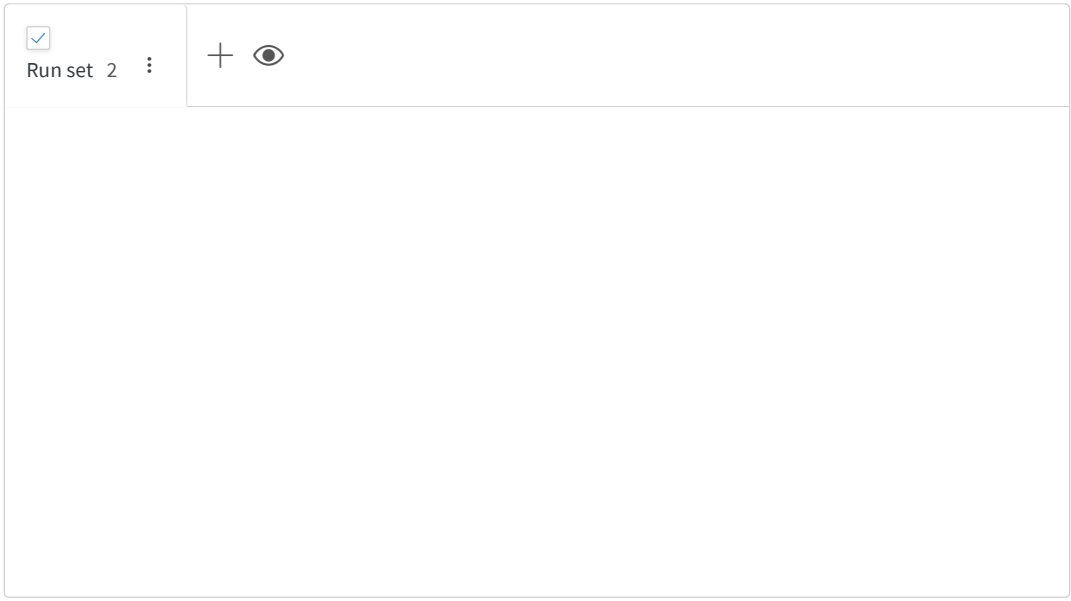
Эксперимент 2

Попробуем увеличить ff_dim до 256. Обучение более стабильное, зазор между лоссами больше, лучшее качество не достигается, опять же, нужно что-то делать с самим обучением. ff_dim 256 будем использовать далее.





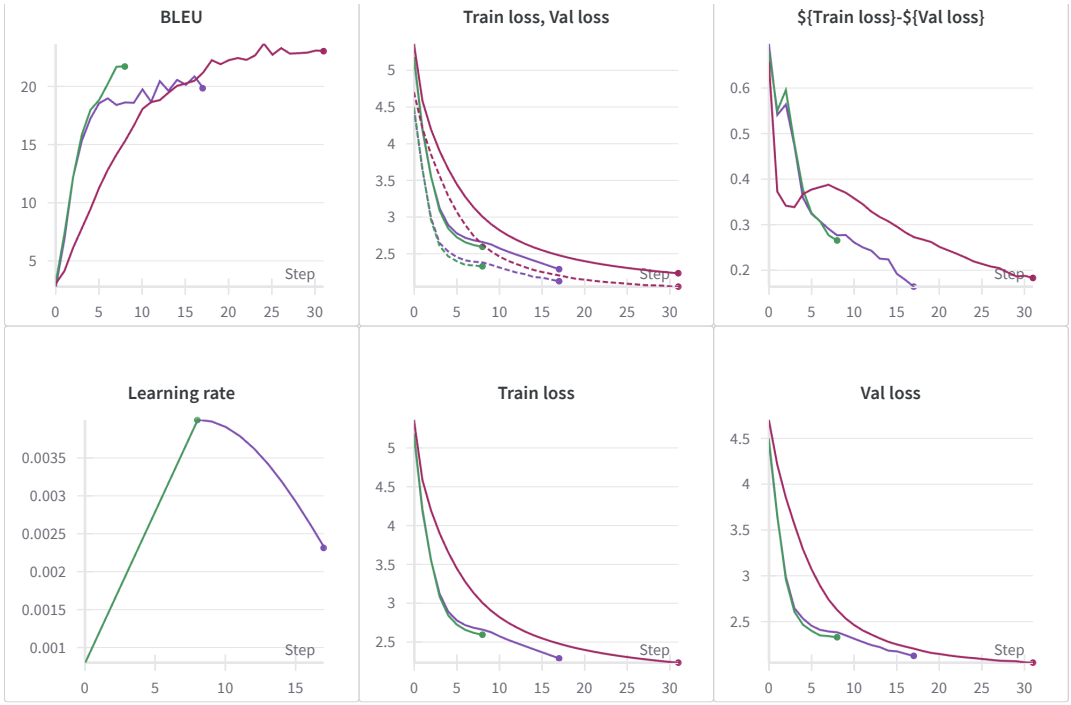
Import panel Add panel



Эксперимент 3

Попробуем добавить расписание обучения. В Attention is All You Need было предложено расписание с линейной warm-up стадией и обратным квадратичным затуханием, но у нас в первой домашке уже написан CosineScheduler, так что возьмём его и будем ориентироваться на меньшие пиковые значения lr, поскольку в них модель будет проводить больше времени из-за особого графика функции.

Сделали два запуска с одинаковыми параметрами (lr в пике 4e-3, разбиение по эпохам 10/20) и видим, что взяли слишком большой lr, из-за чего около его пика может разваливаться обучение, явно видна нестабильность, но даже так на первых эпохах обучение идёт значительно быстрее, чем без расписания.



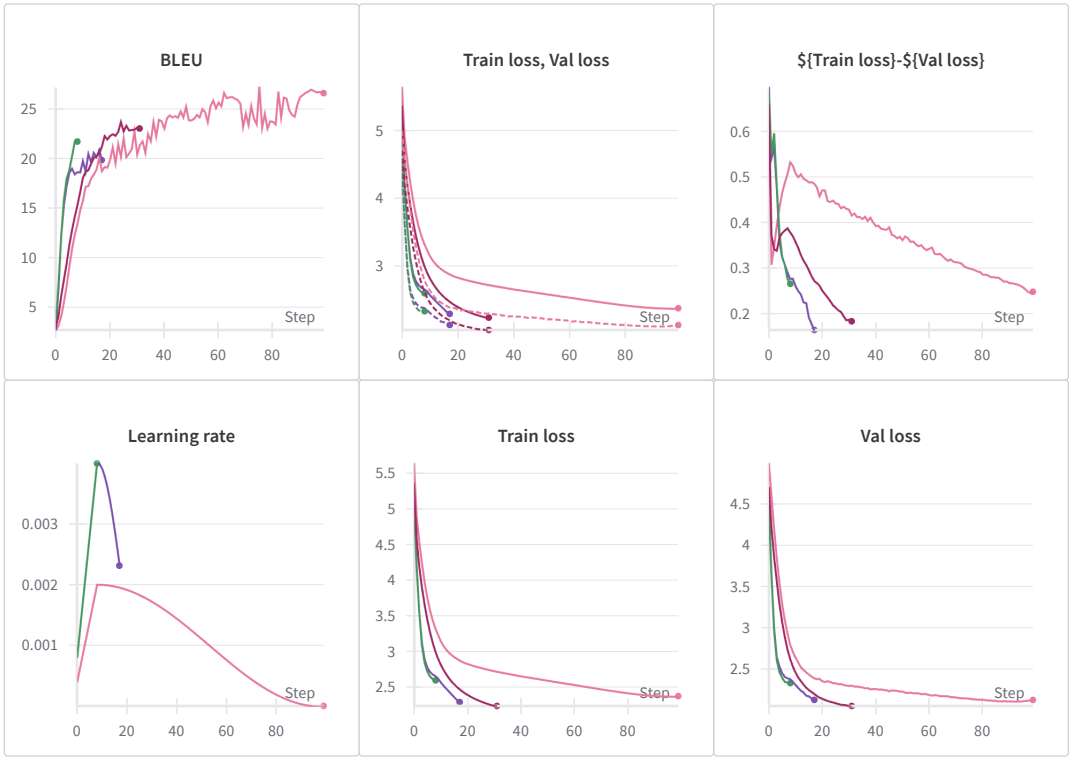
Import panel Add panel

Run set 3

Эксперимент 4

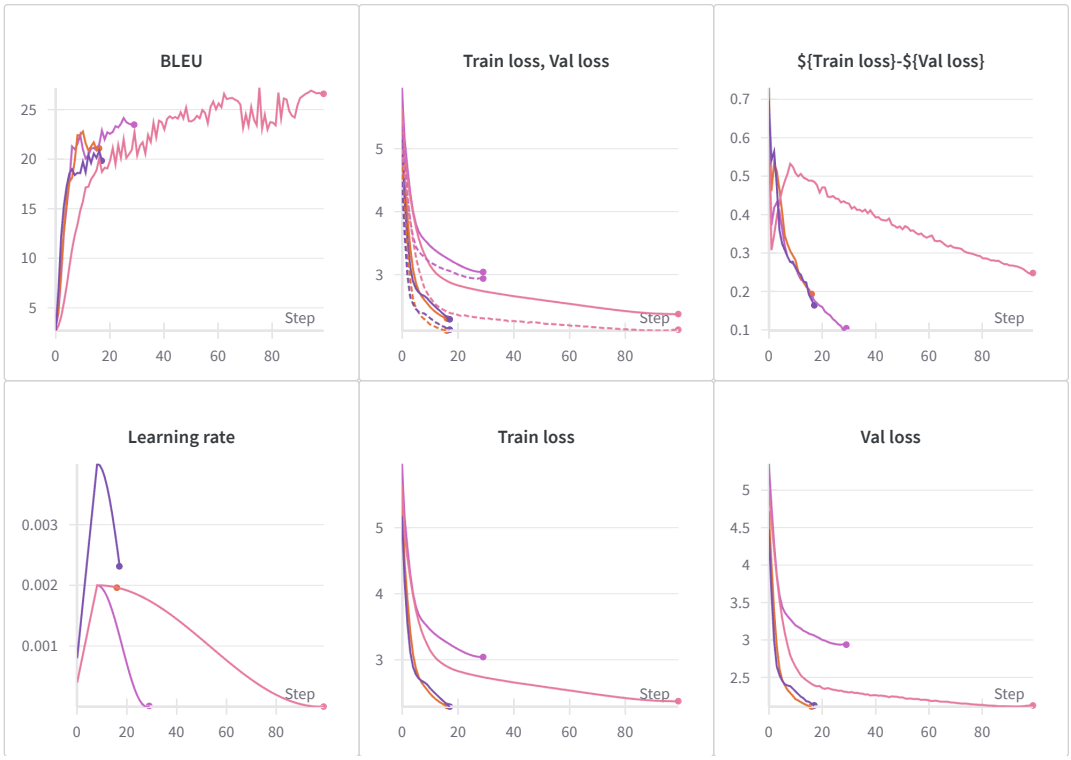
Уменьшим пиковый lr до 2e-3 и поставим dropout=0.3 (увеличив число эпох, поскольку наше обучение будет более медленным), так как переобучение в прошлый раз наступало очень быстро.

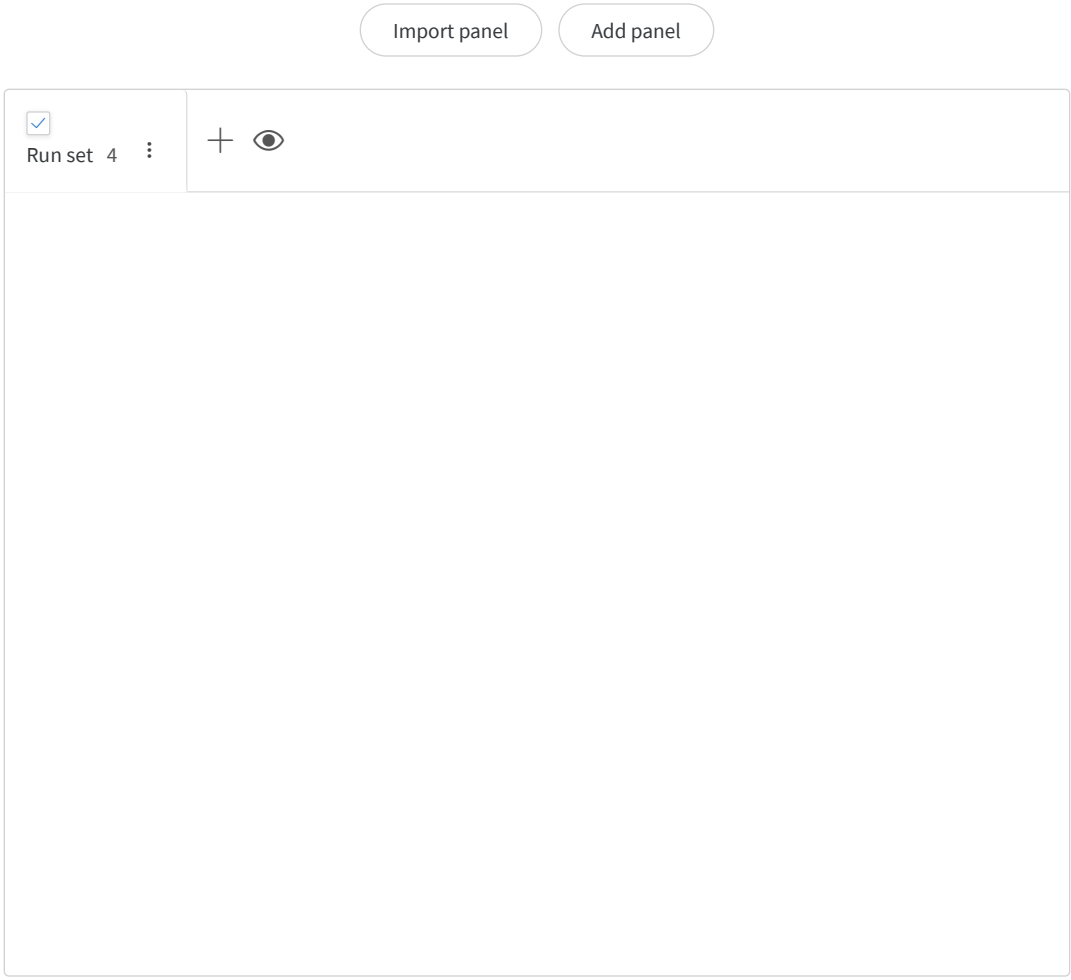
Из-за большого dropout обучение даже чуть медленнее, чем с константным lr, но при сравнении точек обучения с одинаковым BLEU заметна большая разница в зазоре между лоссами. В конце получаем лучший BLEU, но обучение занимает слишком много времени, и даже под конец модель находится в недоученном состоянии.



Попробуем вернуть dropout=0.1, пиковый lr оказался большим для такого расписания.

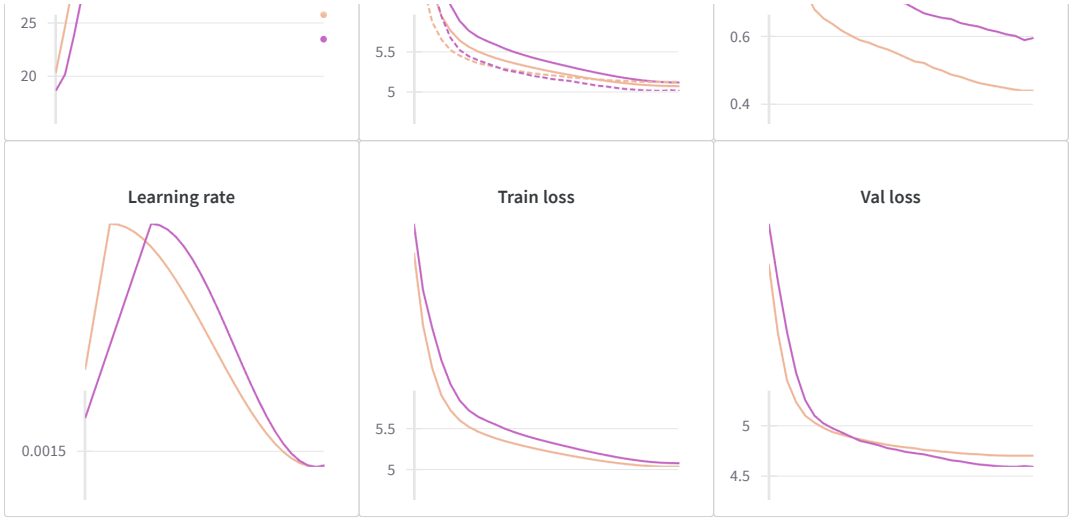
Теперь уменьшим количество эпох до 30 и добавим label smoothing с параметром 0.1. Обучение всё ещё нестабильное, модель долго подвергается большому lr.





Эксперимент 5

Тогда попробуем уменьшить количество эпох в warm-up стадии до 5. Видим прогресс: качество стабильно растёт почти вплоть до конца. Достигли момента, когда модель переобучается, с этим что-то надо делать.

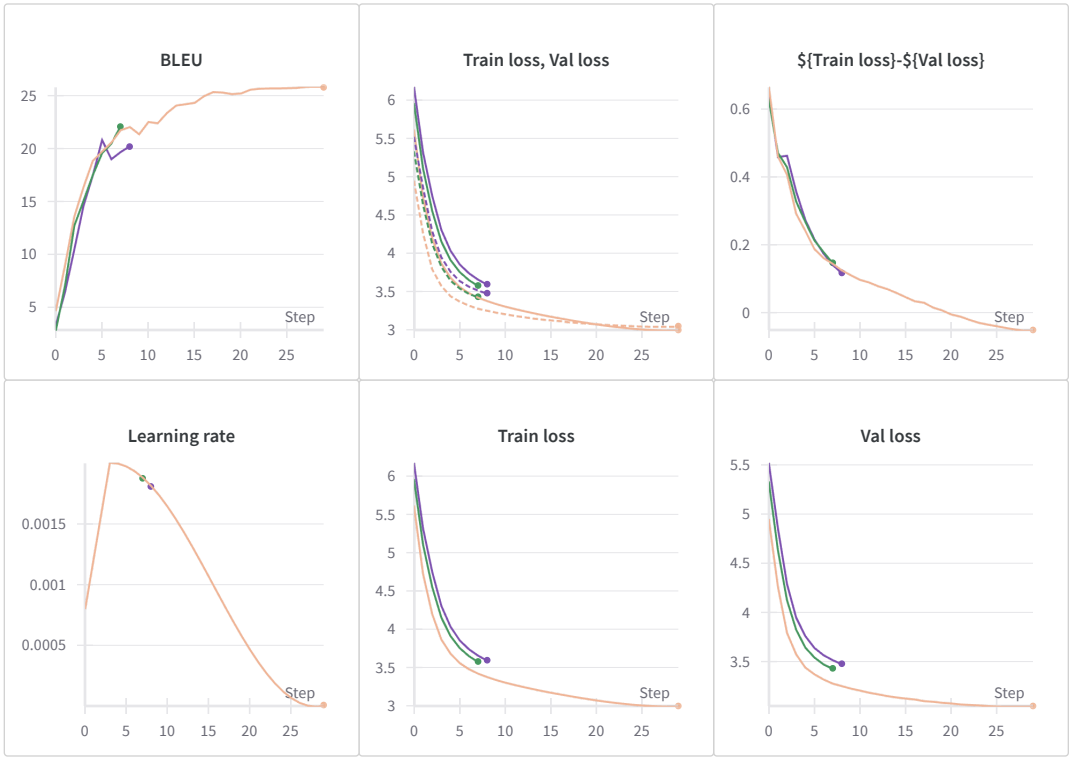


Import panel Add panel

Run set 2

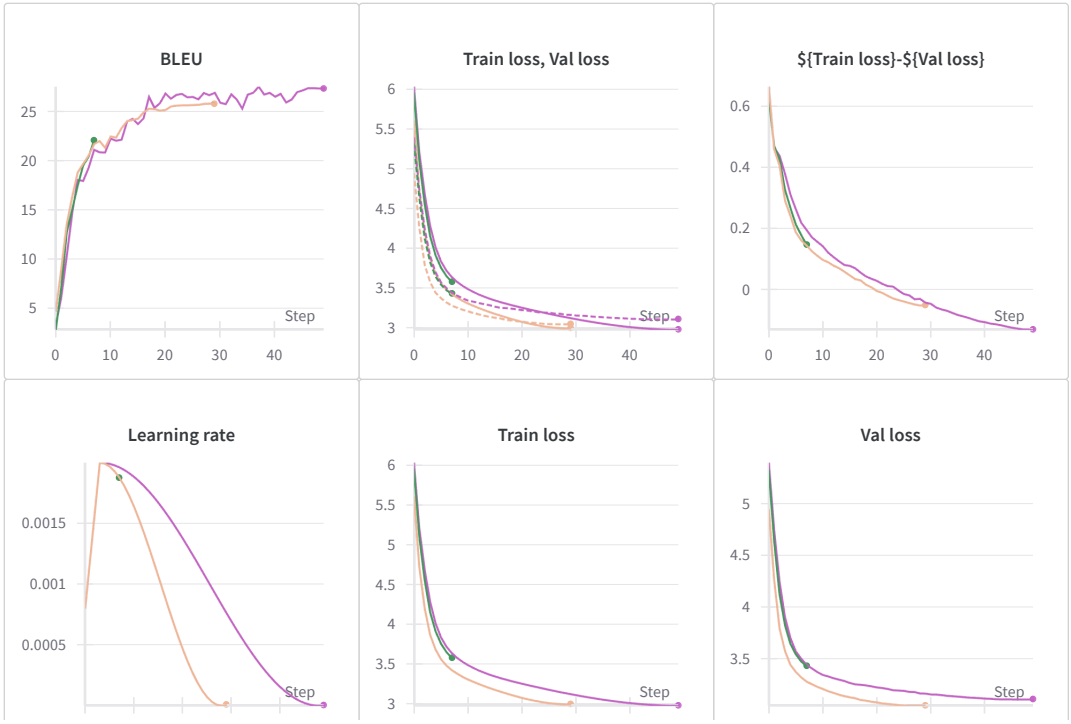
Эксперимент 6

Попробуем немного увеличить модель: уменьшим порог `min_freq` и добавим четвёртый слой энкодеру и декодеру. При уменьшении порога растут лоссы, но зазор между ними быстрее сужается, поэтому нам надо будет ещё докинуть регуляризации для продолжительного запуска.

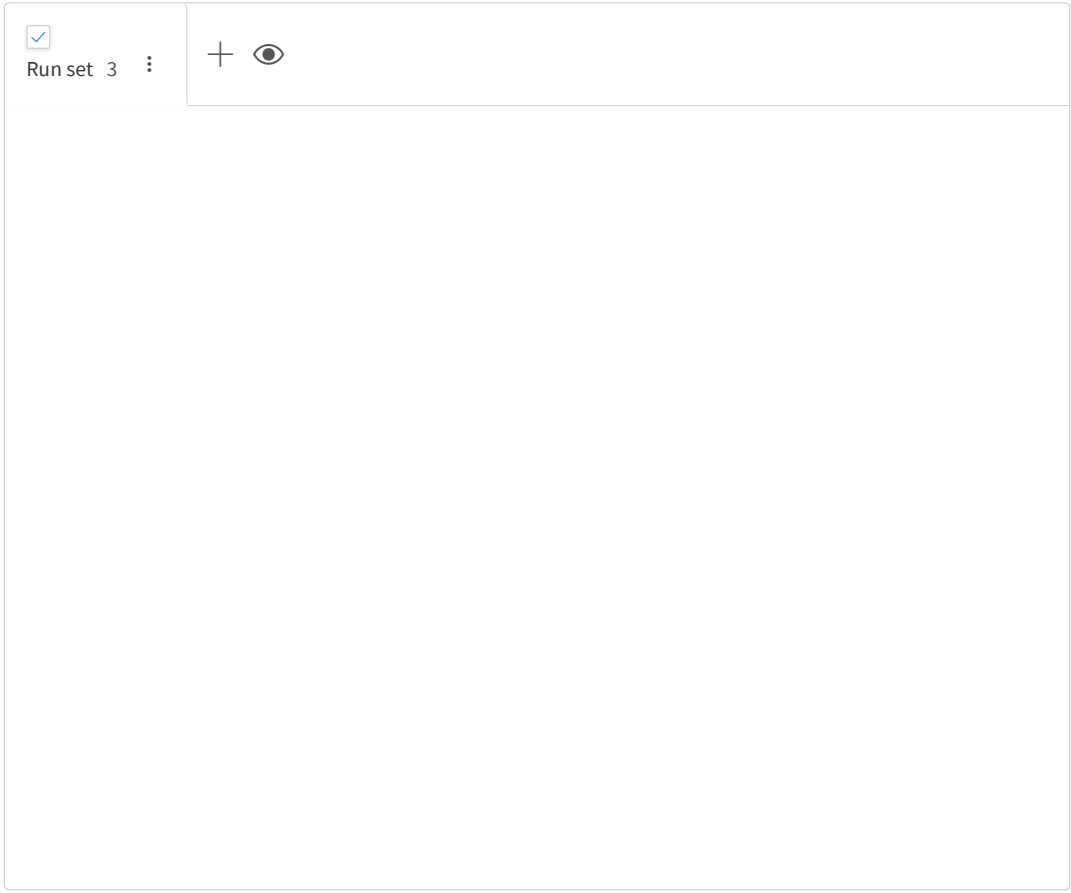


Эксперимент 7

Будем добавлять нормальный шум в эмбеддинги с std=0.2 и запустим обучение на 50 эпох. Видим, что регуляризация работает: обучение происходит немного медленнее, зазор между лоссами сужается даже медленнее, чем у версии с 3 слоями, а общее качество выходит на исторический максимум.

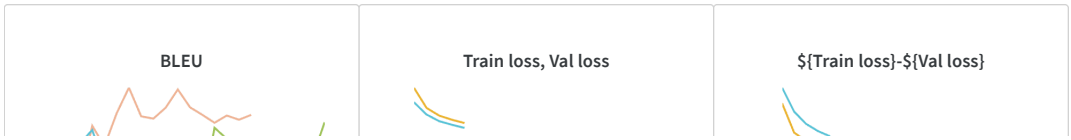


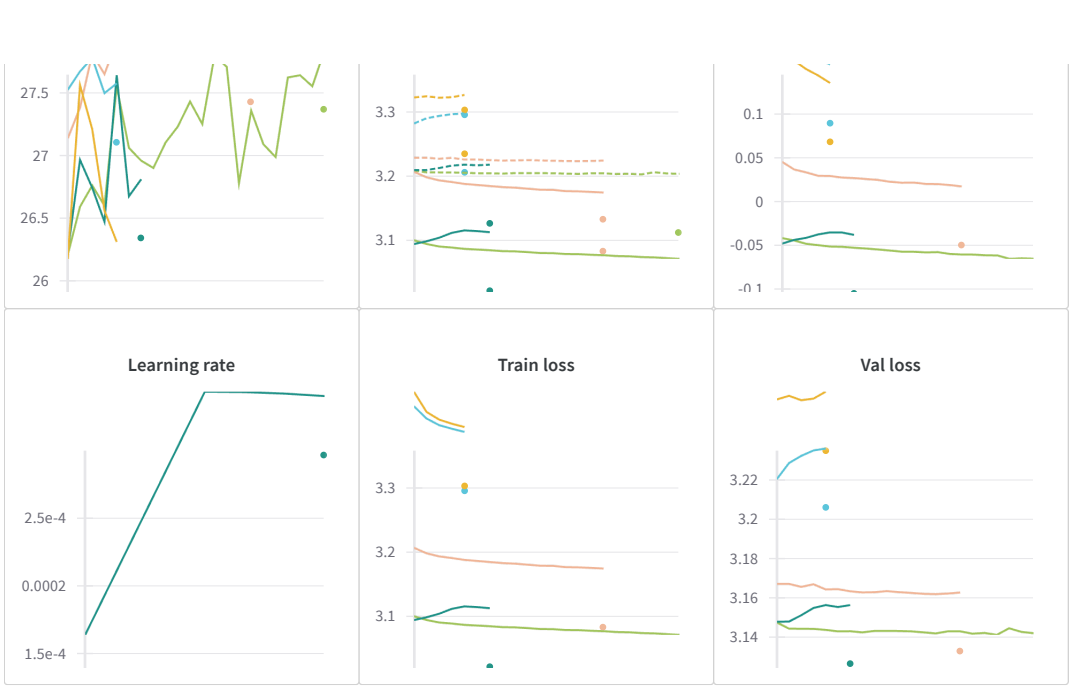
Import panel Add panel



Попытки fine-tuning

Далее я пытался дообучить чекпоинт ed4_mf7_en02, используя разные lr, расписания и регуляризации, но почти везде рос даже лосс на валидации, а BLEU сильно шумел и не улучшался. Возможно, из ed4_mf7_en02_fine_d02_no_sch, в котором использовались Adam с lr=3e-4 и dropout=0.2 можно было вытащить посылку с BLEU лучше того, что был на 50 эпохе ed4_mf7_en02, но кардинальных изменений я не увидел.





Import panel Add panel

Run set 5

Более хороший inference

Конечно, можно было ещё поэкспериментировать и заскейлить модель, в том числе опять уменьшив min_freq, но это требовало бы долгого обучения, да и, как мы уже увидели, трансформеры требовательны к обучению, и изменение архитектуры могло бы потребовать изменений в обучении.

Так что я решил написать beam search.

Реализация, считающая тест за 30-40 минут:

```
# it's really slow, but smh gets the best score
def beam_decode_naive(model, src, src_mask, special_symbols, max_length, beam_topk, all_topk):
    model.eval()
    memory = model.encode(src, src_mask)

    start_token = torch.ones(1, 1).fill_(special_symbols[BOS_STRING]).type(torch.long).to(DEVICE)
    nodes = [(start_token, 0)]
    for _ in range(max_length - 2):
        nodes_new = list()
        all_ended = True
        for seq, score in nodes:
            if seq[0, -1] == special_symbols[EOS_STRING]:
                nodes_new.append((seq, score))
            else:
                all_ended = False
                tgt_mask = generate_square_subsequent_mask(seq.size(1), DEVICE)
                output = model.decode(seq, memory, tgt_mask)
                logits = model.ff(output[:, -1])

                probs = F.log_softmax(logits, dim=1)[0]
                topk_scores, topk_tokens = torch.topk(probs, beam_topk)

                for cur_score, token_num in zip(topk_scores, topk_tokens):
                    token = torch.ones(1, 1).fill_(token_num).type(torch.long).to(DEVICE)
                    nodes_new.append((torch.cat((seq, token), dim=1), score + cur_score))
        if all_ended:
            break
        nodes_new.sort(key=lambda x: x[1], reverse=True)
        nodes = nodes_new[:all_topk]
    return nodes[0][0]
```

Более умная реализация, работающая за 4-6 минут:

```
def beam_decode(model, src, src_mask, special_symbols, max_length, beam_topk, all_topk):
    model.eval()
    memory = model.encode(src.repeat(all_topk, 1), src_mask)

    tgt_batch = torch.ones(all_topk, 1).fill_(special_symbols[BOS_STRING]).type(torch.long).to(DE
    scores = torch.zeros(1, 1).type(torch.float64).to(DEVICE)

    best_end_score = None
    end_seq = None

    end_scores = list()

    for len_i in range(max_length - 2):
        cur_cnt = scores.shape[0]
        nodes_new = list()
        all_ended = True

        tgt_mask = generate_square_subsequent_mask(tgt_batch.size(1), DEVICE)
        output = model.decode(tgt_batch, memory, tgt_mask)
        logits = model.ff(output[:, -1])

        topk_scores, topk_tokens = torch.topk(F.log_softmax(logits[:cur_cnt], dim=1), beam_topk,
        topk_scores += scores

        next_size = min(all_topk, beam_topk * cur_cnt)
        new_scores, raw_top_ind = torch.topk(topk_scores.flatten(), next_size)
        top_ind_i, top_ind_j = raw_top_ind // beam_topk, raw_top_ind % beam_topk

        new_scores = new_scores.unsqueeze(1)

        cur_tokens = topk_tokens[top_ind_i, top_ind_j].unsqueeze(1)
        end_mask = cur_tokens == special_symbols[EOS_STRING]

        new_tokens = torch.ones(all_topk, 1).fill_(special_symbols[BOS_STRING]).type(torch.long).
        new_tokens[:next_size] = topk_tokens[top_ind_i, top_ind_j].unsqueeze(1)

        new_batch = torch.ones(all_topk, len_i + 1).fill_(special_symbols[BOS_STRING]).type(torch
        new_batch[:next_size] = tgt_batch[top_ind_i]

        tgt_batch = torch.cat((new_batch, new_tokens), dim=1)
        scores = new_scores

        end_mask = new_tokens[:next_size] == special_symbols[EOS_STRING]
        if end_mask.any():
```

```
            indices = np.arange(scores.shape[0])[end_mask.flatten().cpu()]
            ind_max = indices[scores[end_mask].flatten().argmax().item()]
            assert end_mask[ind_max]
            end_scores += list(scores[end_mask].flatten())
            end_scores = sorted(end_scores, reverse=True)[:all_topk]
            if best_end_score is None or scores[ind_max] > best_end_score:
                best_end_score = scores[ind_max]
                end_seq = tgt_batch[ind_max].unsqueeze(0)
            scores[end_mask] -= 1e30

        comb_scores = sorted(list(scores.flatten()) + end_scores, reverse=True)
        if len(comb_scores) > all_topk:
            threshold = comb_scores[all_topk]
            scores[scores <= threshold] -= 1e30

        if best_end_score is not None and best_end_score > scores.max():
            break

    ind_scores = scores.argmax()
    res = end_seq if best_end_score is not None and best_end_score > scores[ind_scores] else tgt_
    if res[0, -1] != special_symbols[EOS_STRING]:
        end_token = torch.ones(1, 1).fill_(special_symbols[EOS_STRING]).type(torch.long).to(DEVIC
        res = torch.cat((res, end_token), dim=1)
    return res
```

Возможно, в более быстрой реализации где-то ошибка, но она набирала стабильно более плохой BLEU, даже после нескольких итераций её изменения, когда поведение должно было быть почти полностью аналогично наивной версии.

В целом, beam search себя хорошо показал и дал прирост не более 1 BLEU в зависимости от чекпоинта.

Лучшее решение

Так как же получить посылку с лучшим BLEU?

Обучим ed4_mf7_en02, у неё значимая часть конфига выглядит так:

```
{
  "model": {
      "dropout": 0.1,
      "emb_noise": 0.2,
      "attn_heads": 4,
      "dec_layers": 4,
      "embed_size": 128,
      "enc_layers": 4,
```

```
      "dim_feedforward": 256
  },
  "train": {
      "optim": {
        "adam": {
          "lr": 0.0003,
          "eps": 1e-9,
          "beta1": 0.9,
          "beta2": 0.98,
          "weight_decay": 0
        },
        "name": "Adam"
      },
      "epochs": 50,
      "backend": "gpu",
      "parallel": true,
      "val_size": 0.1,
      "scheduler": {
        "name": "Cosine",
        "cosine": {
          "lr": 0.002,
          "init_lr": 0.000001,
          "decay_lr": 0.00001,
          "decay_epochs": 45,
          "warmup_epochs": 5
        }
      },
      "batch_size": 128,
      "src_min_freq": 7,
      "tgt_min_freq": 7,
      "label_smoothing": 0.1
  }
}
```

(inference с помощью greedy search набирает BLEU 27.5 на public test)

А потом с помощью beam search с параметрами beam_topk=3 и all_topk=8 выполним inference (~~смотря какие fabric какие details~~) и получим BLEU 28.42 на public test. Ура!