Laboratorium 6

RLHF — Reward Model, PPOTrainer i dane preferencji

1. Wprowadzenie

RLHF (Reinforcement Learning from Human Feedback) to technika dostrajania modeli językowych, która pozwala wyrównać model z ludzkimi preferencjami. Zamiast polegać wyłącznie na **supervised learning**, RLHF zbiera pary odpowiedzi ocenione przez człowieka i uczy modelu generowania odpowiedzi, które są oceniane wyżej.

Elementy procesu RLHF omawiane na tych zajęciach:

- **Reward Model** — sieć neuronowa oceniająca jakość odpowiedzi (skalar $r \in \mathbb{R}$).
- **Dane preferencji** — pary (chosen, rejected) z adnotacjami preferencji człowieka.
- **PPOTrainer (TRL)** — pętla RL optymalizująca model językowy przy użyciu reward modelu.
- **Kara KL-divergence** — regularyzacja zapobiegająca reward hackingowi ($\beta \cdot KL$).

> Ważne: Biblioteki takie jak TRL (HuggingFace), OpenRLHF i RL4LMs upraszczają implementację RLHF. Na tych zajęciach korzystamy z biblioteki TRL i modeli DistilBERT oraz distilgpt2 ze względu na wymagania sprzętowe.

2. Przykłady implementacji

Przykład 1: Trenowanie Reward Model na danych preferencji

```
# --- Przyklad 1: Trenowanie Reward Model na danych preferencji ---
from datasets import load_dataset
from transformers import AutoTokenizer, AutoModelForSequenceClassification
from trl import RewardTrainer, RewardConfig

# Zaladuj dane preferencji
dataset = load_dataset('Anthropic/hh-rlhf', split='train[:200]')

# Model i tokenizer
model_name = 'distilbert-base-uncased'
tokenizer  = AutoTokenizer.from_pretrained(model_name)
model      = AutoModelForSequenceClassification.from_pretrained(
    model_name, num_labels=1)

# distilbert nie ma eos_token - uzyjemy pad_token jako pseudo-eos (wymagane przez
trl)
if tokenizer.eos_token is None:
```

```
    tokenizer.eos_token = tokenizer.pad_token

# Tokenizacja par chosen/rejected
def preprocess(example):
    tok_c = tokenizer(example['chosen'],   truncation=True, max_length=256)
    tok_r = tokenizer(example['rejected'], truncation=True, max_length=256)
    return {
        'input_ids_chosen':          tok_c['input_ids'],
        'attention_mask_chosen':     tok_c['attention_mask'],
        'input_ids_rejected':        tok_r['input_ids'],
        'attention_mask_rejected':   tok_r['attention_mask'],
    }

dataset = dataset.map(preprocess, batched=False)

# Trening
config  = RewardConfig(output_dir='./rm_output', num_train_epochs=1,
                       per_device_train_batch_size=4, logging_steps=10,
                       max_length=256,
                       use_cpu=True)
trainer = RewardTrainer(model=model, args=config,
                        processing_class=tokenizer, train_dataset=dataset)
trainer.train()
print('Reward model wytrenowany.')
print('UWAGA: wymaga internetu (pobranie danych Anthropic/hh-rlhf oraz')
print('       modelu distilbert-base-uncased) oraz zainstalowanej biblioteki trl.')
```

Przykład 2: Własna funkcja nagrody (bez TRL)

```
# --- Przyklad 2: Wlasna heurystyczna funkcja nagrody (bez TRL) ---
# Uwaga: usunieto nieuzywane importy (torch/transformers), poniewacz kod ich nie
potrzebuje.

def simple_reward(response: str) -> float:
    # r_len: nagradza umiarkowana dlugosc (do 200 znakow)
    r_len = min(len(response) / 200, 1.0)
    # r_kw:  nagradza uzycie slow kluczowych
    keywords = ['poniewaz', 'dlatego', 'przyklad', 'oznacza']
    r_kw = sum(kw in response.lower() for kw in keywords) / len(keywords)
    # r_uniq: nagradza unikalnosc slow
    words = response.split()
    r_uniq = len(set(words)) / max(len(words), 1)
    return 0.3 * r_len + 0.5 * r_kw + 0.2 * r_uniq

responses = [
    'Tak.',
    'Poniewaz fotosynteza jest procesem biologicznym, oznacza to ze rosliny
produkuja tlen.',
    'Dlatego wynik zalezy od wielu czynnikow, na przyklad temperatury.',
```

```
    'aaaa bbbb cccc dddd eeee',
    'Krotkie zdanie bez slow kluczowych.',
]

print(f"{'Składowe':<60} {'r_len':>6} {'r_kw':>6} {'r_uniq':>7}  Reward")
for r in responses:
    r_len = min(len(r) / 200, 1.0)
    keywords = ['poniewaz', 'dlatego', 'przyklad', 'oznacza']
    r_kw = sum(kw in r.lower() for kw in keywords) / len(keywords)
    words = r.split()
    r_uniq = len(set(words)) / max(len(words), 1)
    total = 0.3 * r_len + 0.5 * r_kw + 0.2 * r_uniq
    print(f'{r[:58]:<60} {r_len:>6.3f} {r_kw:>6.3f} {r_uniq:>7.3f}  {total:.3f}')
```

Przykład 3: Pomiar dywergencji KL między politykami

```
# --- Przyklad 3: Pomiar dywergencji KL miedzy politykami ---
import torch
import torch.nn.functional as F
from transformers import AutoTokenizer, AutoModelForCausalLM

model_name = 'distilgpt2'
tokenizer  = AutoTokenizer.from_pretrained(model_name)

# 'Polityka referencyjna' (SFT) = model bazowy
ref_model    = AutoModelForCausalLM.from_pretrained(model_name)
# 'Polityka uczona' = ta sama architektura, inna temperatura
policy_model = AutoModelForCausalLM.from_pretrained(model_name)
ref_model.eval(); policy_model.eval()

def kl_divergence(prompt: str, temperature: float = 1.5) -> float:
    ids = tokenizer(prompt, return_tensors='pt')['input_ids']
    with torch.no_grad():
        ref_logits    = ref_model(ids).logits[:, -1, :]
        policy_logits = policy_model(ids).logits[:, -1, :] / temperature
    p_ref    = torch.softmax(ref_logits, dim=-1) + 1e-8
    p_ref    = p_ref / p_ref.sum(dim=-1, keepdim=True)
    p_policy = torch.softmax(policy_logits, dim=-1) + 1e-8
    p_policy = p_policy / p_policy.sum(dim=-1, keepdim=True)
    kl = F.kl_div(torch.log(p_ref), p_policy, reduction='sum').item()
    return kl

prompts = ['The capital of France is', 'Machine learning is',
           'The best way to learn Python']
for prompt in prompts:
    kl = kl_divergence(prompt, temperature=1.5)
    print(f'KL={kl:.4f} | {prompt}')
```

Przykład 4: Pętla PPOTrainer (uproszczona)

```
# --- Przyklad 4: Krok treningowy RL (GRPOTrainer w TRL 1.18) ---
# Uwaga: klasyczne PPOTrainer/PPOConfig/AutoModelForCausalLMWithValueHead
# zostaly USUNIETE z TRL 1.x. Tu uzyto nowoczesnego odpowiednika - GRPOTrainer
# z ta sama heurystyczna funkcja nagrody co w przykladzie 2.
from datasets import Dataset
from transformers import AutoTokenizer
from trl import GRPOTrainer, GRPOConfig

model_name = 'distilgpt2'
tokenizer  = AutoTokenizer.from_pretrained(model_name)
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

# Prompty treningowe
prompts = [
    'Explain gravity briefly.',
    'What is the capital of France?',
    'Machine learning is useful because',
]
train_dataset = Dataset.from_dict({'prompt': prompts})

# Heurystyczna funkcja nagrody (bez reward modelu, jak w przykladzie 2)
def reward_fn(prompts, completions, **kwargs):
    scores = []
    for comp in completions:
        text = comp if isinstance(comp, str) else ''.join(comp)
        r_len  = min(len(text) / 200, 1.0)          # umiarkowana dlugosc
        words  = text.split()
        r_uniq = len(set(words)) / max(len(words), 1)  # unikalnosc slow
        scores.append(0.4 * r_len + 0.3 * r_uniq + 0.3)
    return scores

config = GRPOConfig(
    output_dir='./grpo_output',
    num_train_epochs=1,
    max_completion_length=30,
    num_generations=4,
    per_device_train_batch_size=4,
    gradient_accumulation_steps=1,
    learning_rate=5e-5,
    use_cpu=True,
    report_to=[],
)

trainer = GRPOTrainer(
    model=model_name,
    processing_class=tokenizer,
```

```
    args=config,
    train_dataset=train_dataset,
    reward_funcs=[reward_fn],
)
result = trainer.train()
print('\nTrenowanie zakonczone.')
# Wyswietl ostatnie metryki (m.in. nagrode i KL)
for entry in trainer.state.log_history[-6:]:
    clean = {k: (round(float(v), 4) if isinstance(v, (int, float)) else v)
             for k, v in entry.items()}
    print(clean)
```

3. Zadania do samodzielnego rozwiązania

Zadanie 6.1: Własna funkcja nagrody

Zaimplementuj własną funkcję nagrody składającą się z trzech składowych:

(a) r_len — nagroda za umiarkowaną długość odpowiedzi (optymalnie 50–200 znaków, liniowo malejąca poza zakresem),

(b) r_kw — nagroda za użycie słów kluczowych sygnalizujących wyjaśnienie (np. "ponieważ", "dlatego", "przykład", "oznacza"),

(c) r_uniq — nagroda za unikalność słów (stosunek unikalnych słów do wszystkich).

Połącz składowe ze współczynnikami: `reward = 0.3·r_len + 0.5·r_kw + 0.2·r_uniq`.

Przetestuj funkcję na co najmniej 5 odpowiedziach o różnej jakości (bardzo krótka, bardzo długa, idealna, bez słów kluczowych, z dużym powtórzeniami). Wydrukuj wszystkie składowe i wynik końcowy dla każdej odpowiedzi.

Pytania do refleksji: Która odpowiedź dostaje najwyższy wynik? Czy jest to odpowiedź faktycznie najlepsza? Jakie zachowania może "nagradzać" ta funkcja, które byłyby niepożądane w prawdziwym systemie (reward hacking)?

Zadanie 6.2: Trenowanie Reward Model na danych preferencji

Wytrenuj model nagrody na zbiorze Anthropic/hh-rlhf używając biblioteki TRL:

1. Załaduj 250 pierwszych przykładów ze zbioru Anthropic/hh-rlhf (split="train").
2. Użyj modelu distilbert-base-uncased jako bazy (AutoModelForSequenceClassification, num_labels=1).
3. Stokenizuj pary chosen/rejected (max_length=128, truncation=True).
4. Wytrenuj przez 1 epokę używając RewardTrainer z TRL (batch_size=4).
5. Oceń jakość modelu: wygeneruj własne 10 par (chosen, rejected) i sprawdź, czy model nagrody poprawnie rankinguje (r_chosen > r_rejected).

Raport: Ile par (na 10) zostało poprawnie uszeregowanych? Jaka jest średnia różnica r_chosen − r_rejected?

Zadanie 6.3: Analiza kary KL-divergence

Zbadaj wpływ temperatury na dywergencję KL pomiędzy modelem referencyjnym a polityką:

1. Użyj modelu distilgpt2 jako modelu referencyjnego (π_ref).
2. Symuluj "nauczoną politykę" (π_θ) przez skalowanie logitów — zastosuj różne wartości temperatury T ∈ {0.5, 0.8, 1.0, 1.2, 1.5, 2.0}.
3. Dla każdej wartości T oblicz KL(π_θ || π_ref) na zestawie 5 promptów.
4. Oblicz też entropię rozkładu π_θ dla każdego T.

Narysuj wykres: oś X = temperatura T, oś Y = **średnia KL** i **entropia** (dwie linie).

Pytania do refleksji: Przy jakim T polityka najbardziej odbiega od referencji? Jak koreluje entropia z KL? Co oznacza, gdy KL >> 1? Jak odpowiada to roli współczynnika β w PPO?

Zadanie 6.4: Opis procesu RLHF dla własnego przypadku użycia

Zaprojektuj i opisz pełen pipeline RLHF dla wybranego przypadku użycia (np. chatbot obsługi klienta, asystent medyczny, generator dokumentacji kodu, korepetytor językowy).

Opis powinien zawierać:

(a) Cel systemu i dziedzina zastosowania.

(b) Opis danych preferencji: skąd je pobrać / jak zebrać, jakie byłyby kryteria oceny (3 kryteria z uzasadnieniem), możliwe problemy z jakością adnotacji.

(c) Architektura reward modelu: wybór bazowego LLM, rozmiar, uzasadnienie.

(d) Konfiguracja PPOTrainer: wartości hiperparametrów (learning_rate, batch_size, init_kl_coef) z uzasadnieniem każdego wyboru.

(e) Potencjalne przejawy reward hackingu w tym scenariuszu i jak im zapobiec.

Zapisz opis w pliku `rlhf_design.md` (Markdown, min. 400 słów). Możesz dołączyć schemat blokowy (ASCII lub opis tekstowy).

Bonus (+1 pkt): Zaimplementuj zabawkowy (toy) wariant z modelem distilgpt2 i heurystyczną nagrodą dopasowaną do wybranej dziedziny. Uruchom 5 kroków PPOTrainer i zaloguj: mean_reward i objective/kl po każdym kroku.

4. Przydatne zasoby i biblioteki

Zasób	Opis	Link/pakiet
TRL (HuggingFace)	RewardTrainer, PPOTrainer, DPOTrainer	`pip install trl`

Zasób	Opis	Link/pakiet
Anthropic/hh-rlhf	Publicznie dostępne dane preferencji	HuggingFace Datasets
InstructGPT paper	Oryginalny artykuł RLHF (Ouyang 2022)	arxiv.org/abs/2203.02155
Zephyr paper	SFT + DPO alternatywa dla RLHF	arxiv.org/abs/2310.16944
TRL docs	Dokumentacja PPOTrainer i przykłady	huggingface.co/docs/trl

Wymagania techniczne

Wymagane wersje bibliotek (potwierdzone; środowisko Python 3.11 / Windows 8.1/10/11):
- transformers == 5.3.0 (przy braku GPU uzywaj use_cpu=True)
- torch == 2.8.0
- datasets == 4.6.1 (NIE stosuj 5.x - wymaga nowszego pyarrow/pa.json_())
- pyarrow == 16.1.0 (obowiazkowe przypiecie na Win 8.1/10/11; nowsze sie nie laduja)
- accelerate >= 1.1.0
- trl == 1.18.0 (PPOTrainer usuniety -> RewardTrainer / GRPOTrainer)

Instalacja:
pip install "transformers==5.3.0" "torch==2.8.0" "datasets==4.6.1" "pyarrow==16.1.0" "accelerate>=1.1.0" "trl==1.18.0"

Modele: distilbert-base-uncased (Reward Model), distilgpt2 (policy)
Dataset: Anthropic/hh-rlhf (HuggingFace Hub, dostep publiczny)