3. vježba: analiza klasifikacije sentimenta

U trećoj laboratorijskoj vježbi bavimo se problemom analize sentimenta recenzija filmova pomoću povratnih neuronskih mreža. Kao skup podataka analizirati ćemo Stanford Sentiment Treebank (SST), manji skup podataka koji je često korišten u obradi prirodnog jezika. SST je specifičan jer u svom punom obliku sadrži anotacije vrijednosti sentimenta na svakom čvoru u stablu parsiranja svake instance. Nažalost, usprkos iscrpnoj anotaciji dokumenata, taj skup podataka je često kritiziran kao pretjerano jednostavan budući da se za većinu instanci klasifikacija svodi na prepoznavanje ključnih riječi.

Skup podataka **nemojte** skidati sa službenog repozitorija budući da je zapis u stablastom formatu. Uz vježbu su priložene predprocesirane verzije skupa podataka gdje su podaci pretvoreni u lowercase, a neki tokeni su filtrirani. Osim skupa podataka, za potrebe laboratorijske vježbe dobiti ćete i set vektorskih reprezentacija za sve riječi koje su prisutne u **train** splitu SST dataseta.

Train, test i validacijski skup podataka za SST možete naći ovdje. Vektorske reprezentacije možete preuzeti ovdje.

Vaš zadatak u trećoj laboratorijskoj vježbi je provjeriti koliko je SST stvarno jednostavan skup podataka te evaluirati povratne neuronske mreže kao i alternativne pristupe bazirane na sažimanju. Kroz pripremu postoji niz kontrolnih ispisa, koji bi trebali biti isti u vašoj implementaciji osim ako nije navedeno drukčije.

Pogled na skup podataka: Stanford Sentiment Treebank

Podaci u tekstnim datotekama odvojeni su s zarezom i jednim razmakom: `, `. Zarez se neće pojavljivati unutar podataka. Podaci se sastoje od teksta ulaznog primjera i ciljne oznake razreda. Oznaka razreda je `positive` ili `negative`, dok je tekst niz predsegmentiranih tokena odvojenih jednim znakom razmaka. Tekst ulaznog primjera i oznaka razreda su nešto što zovemo **poljima** (engl. field). Primjetite da je značenje polja ovdje u kontekstu “polje za unos podataka”, a ne obrađena travnata površina. Primjer prve instance skupa za testiranje:

```
it 's a lovely film with lovely performances by buy and accorsi, positive
```

Podaci iz SST-a se nalaze u tri datoteke - `sst_train_raw.csv`, `sst_valid_raw.csv`, `sst_test_raw.csv`. Sve tri datoteke su u istom formatu.

Zadatak 1. Učitavanje podataka (25% bodova)

Za razliku od računalnog vida i `torchvision` paketa, ekosustav za obradu prirodnog jezika je veoma fragmentiran i postoje razne konfliktne ideologije učitavanja podataka. Više biblioteka za ove svrhe postoji (`torchtext`, `AllenNLP`, …) no njihove funkcionalnosti u okviru vježbe nećemo koristiti.

Kako bi bolje shvatili problematiku kojom se susrećemo s analizom teksta, vaš prvi zadatak je implementirati učitavanje skupa podataka s diska. Za iteriranje i batchiranje podataka ćemo koristiti razrede ugrađene u

Pytorch (`torch.utils.data.DataLoader` i `torch.utils.data.Dataset`). Pri učitavanju podataka predlažemo vam da koristite konceptualnu podjelu na iduća tri razreda, čije se imenovanje **smije razlikovati**:

- Razred `Instance`, koji je plitki omotač oko podataka. Jednostavni i korisni načini implementacije ovakvog razreda su razredi podataka, dostupne od python verzije 3.7. Alternativa su imenovane ntorke.
- Razred `NLPDataset`, koji treba nasljeđivati `torch.utils.data.Dataset` te implementirati `__getitem__` metodu. Ovaj razred služi za spremanje i dohvaćanje podataka te operacije za koje nam je potreban cijeli skup podataka, poput izgradnje vokabulara.
- Razred `Vocab`, koji nam služi za pretvorbu tekstnih podataka u indekse (što zovemo *numerikalizacija*). Funkcionalnost razreda Vocab ćemo detaljnije analizirati u idućem podpoglavju.

Razred `Vocab`

Kao što smo spomenuli na predavanjima, jedan od hiperparametara svakog modela obrade prirodnog jezika je odabir veličine vokabulara, tj., broja riječi koje ćemo reprezentirati u našem modelu. Procedura odabira se u praksi provodi u nekoj vrsti `Vocab` razreda, pri izgradnji rječnika `itos` (index-to-string) i `stoi` (string-to-index).

Svakom ulaznom polju našeg skupa podataka pridodjeljujemo jedan vokabular. Vaša implementacija vokabulara se treba izgraditi temeljem rječnika frekvencija za neko polje. Rječnik frekvencija kao ključeve sadrži sve tokene koji su se pojavili u tom polju, dok su vrijednosti broj pojavljivanja svakog tokena.

Primjer 5 najčešćih riječi i njihovih frekvencija za polje ulaznog teksta `train` skupa podataka, koje sadrži `14804` različitih tokena:

```
the, 5954
a, 4361
and, 3831
of, 3631
to, 2438
```

Konvencija je češćim riječima pridodjeljivati niže indekse. No, prije nego krenemo s pretvorbom riječi u indekse, trebamo se pozabaviti s idejom posebnih znakova (`special symbols`).

Posebni znakovi su tokeni koji se **ne** nalaze u našem skupu podataka ali su nužni našem modelu za numerikalizaciju podataka. Primjeri ovih znakova koje ćemo koristiti u laboratorijskoj vježbi su znak punjenja `<PAD>` (eng. padding) i nepoznati znak `<UNK>`. Znak punjenja nam je potreban kako bi naše batcheve (koji se sastoje od primjera različite duljine) sveli na jednaku duljinu, dok nam nepoznati znak služi za riječi koje nisu u našem vokabularu – bilo zbog ograničenja veličine ili jer se nisu pojavile dovoljno puta u podacima.

Posebni znakovi uvijek imaju najniže indekse, a radi konzistentnosti s praksom ćemo znaku punjenja uvijek dati indeks 0, a nepoznatom znaku indeks 1. Preostalim rječima se indeksi trebaju dodijeliti prema njihovoj frekvenciji po principu da riječ koja se češće pojavljuje ima niži indeks. Posebni znakovi se koriste **samo u tekstnom polju**.

Primjer riječi i njihovih indeksa u našem `stoi` rječniku vokabulara polja ulaznog teksta:

```
<PAD>: 0
<UNK>: 1
the: 2
```

```
a: 3
and: 4
my: 188
twists: 930
lets: 956
sports: 1275
amateurishly: 6818
```

Dok je `stoi` rječnik za vokabular polja ciljne varijable dosta kratak:

```
positive, 0
negative, 1
```

Vaša implementacija razreda `Vocab` mora implementirati funkcionalnost pretvorbe niza tokena (ili jednog tokena) u brojeve. Ovu funkciju možete nazvati `encode`. Primjer ove pretvorbe za četvrtu instancu train skupa:

```
instance_text, instance_label = train_dataset.instances[3]
print(f"Text: {instance_text}")
print(f"Label: {instance_label}")
>>> Text: ['yet', 'the', 'act', 'is', 'still', 'charming', 'here']
>>> Label: positive
print(f"Numericalized text: {text_vocab.encode(instance_text)}")
print(f"Numericalized label: {label_vocab.encode(instance_label)}")
>>> Numericalized text: tensor([189,   2, 674,   7, 129, 348, 143])
>>> Numericalized label: tensor(0)
```

Također, vaša implementacija razreda `Vocab` mora primati iduće parametre:

- `max_size`: maksimalni broj tokena koji se sprema u vokabular (uključuje i posebne znakove). `-1` označava da se spremaju svi tokeni.
- `min_freq`: minimalna frekvencija koju token mora imati da bi ga se spremilo u vokabular (\ge). Posebni znakovi ne prolaze ovu provjeru.

Primjer izgradnje vokabulara sa svim tokenima (duljina uključuje i posebne znakove):

```
text_vocab = Vocab(frequencies, max_size=-1, min_freq=0)
print(len(text_vocab.itos))
14806
```

Bitno: vokabular se izgrađuje **samo** na train skupu podataka. Jednom izgrađeni vokabular na train skupu postavljate kao vokabular testnog i validacijskog skupa podataka. Ovaj pristup se smatra najkorektniji u analizi teksta jer kroz izgradnju vokabulara na testnom i validacijskom skupu imamo curenje informacija u treniranje modela. Primjerice – u realnoj situaciji gdje deployate vaš model nije vjerojatno da će vaš model svaku viđenu riječ imati u vokabularu. Prema tome, ovakav način evaluacije, iako stroži, je realističniji.

Učitavanje vektorskih reprezentacija

U okviru laboratorijske vježbe, uz skup podataka, dobiti ćete i podskup prednaučenih reprezentacija riječi GloVe. Ove vektorske reprezentacije možete preuzeti ovdje.

Vektorske reprezentacije riječi su zapisane u tekstnom formatu, pri čemu svaki redak sadrži token (riječ) i njenu 300-dimenzijsku vektorsku reprezentaciju. Vektorske reprezentacije koje koristimo u vježbi će uvijek biti 300-dimenzijske. Elementi svakog retka su odvojeni jednim zarezom.

Primjer prvog retka:

```
the 0.04656 0.21318 -0.0074364 -0.45854 -0.035639 ...
```

Vaš zadatak je implementirati funkciju koja će za zadani vokabular (iterable stringova) generirati embedding matricu. Vaša funkcija treba podržavati dva načina genriranja embedding matrice: nasumična inicijalizacija iz standardne normalne razdiobe $(\mathbb{N}(0,1))$ i učitavanjem iz datoteke. Pri učitavanju iz datoteke, ako ne pronađete vektorsku reprezentaciju za neku riječ, inicijalizirajte ju normalno. Vektorsku reprezentaciju za znak punjenja (na indeksu 0) **morate** inicijalizirati na vektor nula. Jednostavan način na koji možete implementirati ovo učitavanje je da inicijalirate matricu iz standardne normalne razdiobe, a potom prebrišete inicijalnu reprezentaciju u retku za svaku riječ koju učitate. **Bitno:** Pripazite da redoslijed vektorskih reprezentacija u matrici odgovara redoslijedu riječi u vokabularu! Npr., na indeksu 0 mora biti reprezentacija za posebni znak punjenja.

Jednom kad ste uspješno učitali vašu $V \times d$ embedding matricu, iskoristite `torch.nn.Embedding.from_pretrained()` kako bi vašu matricu spremili u optimizirani omotač za vektorske reprezentacije. Postavite parametar funkcije `padding_idx` na 0 (indeks znaka punjenja u vašoj embedding matrici), a parametar funkcije `freeze` ostavite na `True` ako koristite predtrenirane reprezentacije, a postavite na `False` inače.

Nadjačavanje metoda `torch.utils.data.Dataset`

Da bi naša implementacija razreda `NLPDataset` bila potpuna, potrebno je nadjačati `__getitem__` metodu koja omogućava indeksiranje razreda. Za potrebe vježbe, ta metoda treba vraćati numerikalizirani text i labelu referencirane instance. Također, dovoljno je napraviti da se numerikalizacija radi “on-the-fly”, i nije ju nužno cachirati.

Primjer numerikalizacije s nadjačavanjem:

```
instance_text, instance_label = train_dataset.instances[3]
# Referenciramo atribut klase pa se ne zove nadjačana metoda
print(f"Text: {instance_text}")
print(f"Label: {instance_label}")
>>> Text: ['yet', 'the', 'act', 'is', 'still', 'charming', 'here']
>>> Label: positive
numericalized_text, numericalized_label = train_dataset[3]
# Koristimo nadjačanu metodu indeksiranja
print(f"Numericalized text: {numericalized_text}")
print(f"Numericalized label: {numericalized_abel}")
>>> Numericalized text: tensor([189,   2, 674,   7, 129, 348, 143])
>>> Numericalized label: tensor(0)
```

Implementacija batchiranja podataka: `collate` funkcija

Skoro smo spremni za implementaciju modela – jedino što je preostalo je implementacija pretvorbi niza elemenata u batch podataka. Ovdje se ponovno susrećemo s problemom varijabilnosti dimenzije.

Pytorchev `torch.utils.data.DataLoader` pri spremanju podataka u batcheve u svojoj defaultnoj implementaciji collate funkcije očekuje da su elementi batcha jednake duljine. Ovo u tekstu nije slučaj, te u praksi moramo implementirati vlastitu collate funkciju.

Prvo ćemo definirati što je uopće collate funkcija. Detaljna dokumentacija može se naći ovdje, dok je grubo objašnjenje da se collate funkcija koristi za izgradnju batch tenzora za svako ulazno polje liste instanci. Primjer kostura te metode:

```
def collate_fn(batch):
    """
    Arguments:
      Batch:
        list of Instances returned by `Dataset.__getitem__`.
    Returns:
      A tensor representing the input batch.
    """

    texts, labels = zip(*batch) # Assuming the instance is in tuple-like form
    lengths = torch.tensor([len(text) for text in texts]) # Needed for later
    # Process the text instances
    return texts, labels, lengths
```

Bitno: u vašoj collate funkciji vraćajte i duljine originalnih instanci (koje nisu nadopunjene). Duljine će nam poslužiti pri implementaciji naših modela.

Zadatak naše collate funkcije biti će nadopuniti duljine instanci znakom punjenja do duljine najdulje instance u batchu. Za ovo, pogledajte funkciju `from torch.nn.utils.rnn.pad_sequence`. Primjetite da vaša implementacija collate funkcije mora znati koji se indeks koristi kao znak punjenja.

Jednom kad smo implementirali sve navedeno, naše učitavanje podataka bi moglo izgledati ovako:

```
def pad_collate_fn(batch, pad_index=0):
    #...
    pass
batch_size = 2 # Only for demonstrative purposes
shuffle = False # Only for demonstrative purposes
train_dataset = NLPDataset.from_file('data/sst_train_raw.csv')
train_dataloader = DataLoader(dataset=train_dataset, batch_size=batch_size,
                              shuffle=shuffle, collate_fn=pad_collate_fn)
texts, labels, lengths = next(iter(train_data_loader))
print(f"Texts: {texts}")
print(f"Labels: {labels}")
print(f"Lengths: {lengths}")
>>> Texts: tensor([[   2,  554,    7, 2872,    6,   22,    2, 2873, 1236,    8,   96,
                       4,   10,   72,    8,  242,    6,   75,    3, 3576,   56, 3577,
                    2022, 2874, 7123, 3578, 7124,   42,  779, 7125,    0,    0],
```

```
                  [   2, 2875, 2023, 4801,    5,    2, 3579,    5,    2, 2876, 4802,
                     40,  829,   10,    3, 4803,    5,  627,   62,   27, 2877, 2024,
                    962,  715,    8, 7126,  555,    5, 7127, 4805,    8, 7128]])
>>> Labels: tensor([0, 0])
>>> Lengths: tensor([32, 34])
```

U stvarnoj implementaciji postavite veličinu batcha na veći broj i zastavicu shuffle na `True` za podatke skupa za učenje.

Zadatak 2. Implementacija baseline modela (25% bodova)

Prvi korak kod svakog zadatka strojnog učenja bi trebao biti implementacija baseline modela. Baseline model nam služi za procjenu performansi koje naš stvarni, uobičajeno *skuplji* model mora moći preći kao plitak potok. Također, baseline modeli će nam pokazati kolika je stvarno cijena izvođenja naprednijih modela.

Vaš zadatak u laboratorijskoj vježbi je implementirati model koji će koristiti sažimanje usrednjavanjem (*eng. mean pooling*) kako bi eliminirao problematičnu varijabilnu dimenziju. Pri primjeni sažimanja usrednjavanjem odmah eliminirajte **cijelu** vremensku dimenziju (tzv. *okno* je veličine T).

Osnovni model koji implementirate mora izgledati ovako:

```
avg_pool() -> fc(300, 150) -> ReLU() -> fc(150, 150) -> ReLU() -> fc(150,1)
```

Kao gubitak predlažemo da koristite BCEWithLogitsLoss, u kojem slučaju ne morate primjeniti sigmoidu na izlaznim logitima. Alternativno, možete staviti da vam je izlazna dimenzionalnost broj klasa te koristiti gubitak unakrsne entropije. Oba pristupa su korištena u praksi ovisno o osobnim preferencama.

Kao algoritam optimizacije koristite Adam.

Implementirajte metrike praćenja performansi modela. Osim gubitka na skupu podataka, zanimaju nas **preciznost** (*eng. accuracy*), **f1 mjera** i **matrica zabune** (*eng. confusion matrix*). Nakon svake epohe ispišite performanse modela po svim metrikama na skupu za validaciju, a nakon zadnje epohe ispišite performanse modela na skupu za testiranje.

Radi usporedbe, naša implementacija osnovnog modela za vokabular koji koristi sve riječi (`max_size=-1, min_freq=1`) te inicijalizira njihove reprezentacije s predtreniranima, `seed=7052020`, `lr=1e-4`, `batch_size=10` na skupu za treniranje i `batch_size=32` na skupovima za validaciju i testiranje ostvaruje iduću preciznost:

```
Epoch 1: valid accuracy = 64.031
Epoch 2: valid accuracy = 66.941
Epoch 3: valid accuracy = 72.268
Epoch 4: Valid accuracy = 75.563
Epoch 5: Valid accuracy = 78.199

Test accuracy = 77.646
```

Postavljanje random seeda za pytorch operacije na CPU se vrši sa `torch.manual_seed(seed)`, dok istu stvar napravite i ukoliko u vašem kodu koristite numpy sa `np.random.seed(seed)`. Ako pokrećete kod na grafičkoj kartici, obratite pozornost na upozorenja ovdje. Povratne neuronske mreže su CUDNN optimizirane, te je moguće da reproducibilnost nije 100% osigurana osim ako ne pratite upute s poveznice nauštrb brzine.

Bitno: Dok god rezultati vašeg koda ne variraju iznimno puno (za različita pokretanja), točne izlazne brojke ne moraju biti savršeno jednake. Kako bi provjerili varijancu (tj. stabilnost) vašeg modela, vaš konačni model pokrenite barem **5** puta s istim hiperparametrima, ali različitim seedom. Zapišite (u excel tablicu, word dokument ili slično) rezultate izvođenja (sve navedene metrike) za svaki seed. U komentar dodajte i hiperparametre za pokretanje modela.

Organizacija koda za modele implementirane u Pytorchu

Zbog “syntactic sugara” koji prati treniranje i evaluaciju Pytorch modela, implementacija treniranja modela se dosta često odvaja u tri semantičke cjeline:

1. Inicijalizacija
    - Parsiranje argumenata
    - Učitavanje podataka
    - Inicijalizacija mreže
2. Petlja za treniranje
    - `train` metoda koja izvršava jednu epohu na skupu za treniranje
    - Syntactic sugar:
        - `model.train()` - omogućava dropout
        - `model.zero_grad()` za svaki batch - brisanje prethodnih gradijenata na parametre se ne provodi automatski
        - `loss.backward()` -propagacija lossa na parametre
        - **[Opcionalno]** `torch.nn.utils.clip_grad_norm_(model.parameters(), clip_value)` - podrezivanje gradijenata po normi
        - `optimizer.step()` - ažuriranje parametara na temelju optimizacijskog algoritma i vrijednosti gradijenata
3. Petlja za evaluaciju
    - `evaluate` metoda koja izvrcd šava jednu epohu na skupu za validaciju ili testiranje
    - Syntactic sugar:
        - `with torch.no_grad():` - gradijenti se ne prate (memorijska i vremenska efikasnost)
        - `model.eval()` - onemogućava dropout

Konkretni kosturi ovih metoda mogli bi izgledati kao u nastavku:

```
def train(model, data, optimizer, criterion, args):
  model.train()
  for batch_num, batch in enumerate(data):
    model.zero_grad()
    # ...
    logits = model(x)
    loss = criterion(logits, y)
    loss.backward()
    torch.nn.utils.clip_grad_norm_(model.parameters(), args.clip)
    optimizer.step()
    # ...
```

```
def evaluate(model, data, criterion, args):
  model.eval()
  with torch.no_grad():
    for batch_num, batch in enumerate(data):
      # ...
      logits = model(x)
      loss = criterion(logits, y)
      # ...

def main(args):
  seed = args.seed
  np.random.seed(seed)
  torch.manual_seed(seed)

  train_dataset, valid_dataset, test_dataset = load_dataset(...)
  model = initialize_model(args, ...)

  criterion = nn.BCEWithLogitsLoss()
  optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)

  for epoch in range(args.epochs):
    train(...)
    evaluate(...)
```

Zadatak 3. Implementacija povratne neuronske mreže (25% bodova)

Nakon što ste uspješno implementirali vaš baseline model, vrijeme je da isprobamo neki model baziran na povratnim neuronskim mrežama. Vaš zadatak je implementirati osnovni model povratne neuronske meže **po izboru**. Na izboru su vam iduće ćelije: ["Vanilla" RNN, GRU, LSTM].

Za odabrani model, detaljno pročitajte njegovu dokumentaciju. U nastavku ćemo vam samo skrenuti pozornost na nekoliko bitnih detalja:

- Svaka RNN mreža kao izlaz svoje `forward` metode vraća (1) niz skrivenih stanja posljednjeg sloja i (2) skriveno stanje (tj., skrivena stanja u slučaju LSTMa) za sve slojeve zadnjeg vremenskog koraka. Kao ulaz u dekoder obično želite staviti skriveno stanje iz zadnjeg sloja u zadnjem vremenskom koraku. Kod LSTMa, to je `h` komponenta dualnog `(h, c)` skrivenog stanja.
- Radi brzine, RNN mreže preferiraju inpute u `time-first` formatu (budući da je brže *iterirati* po prvoj dimenziji tenzora). Transponirajte ulaze prije nego ih šaljete RNN ćeliji.
- Tenzori koji su ulaz u RNN ćelije se često "pakiraju". Pakiranje je zapis tenzora kojemu su pridružene stvarne duljine svakog elementa u batchu. Ako koristite pakiranje, RNN mreža se neće odmatati za vremenske korake koji sadrže padding u elementima batcha. Ovdje osim efikasnosti možete dobiti i na preciznosti, ali ovaj dio **nije** nužan dio vaše implementacije.
- Implementirajte gradient clipping prije optimizacijskog koraka Osnovni model vaše odabrane RNN ćelije treba izgledati ovako:

```
rnn(150) -> rnn(150) -> fc(150, 150) -> ReLU() -> fc(150,1)
```

Vaš osnovni model RNN ćelije bi trebao biti jednosmjeran i imati dva sloja. Za višeslojni RNN iskoristite argument `num_layers` pri konstrukciji RNN mreže.

Radi usporedbe, naša implementacija GRU povratne mreže za vokabular koji koristi sve riječi (`max_size=-1, min_freq=1`) te inicijalizira njihove reprezentacije s predtreniranima, `seed=7052020`, `lr=1e-4`, `batch_size=10`, `gradient_clip=0.25` na skupu za treniranje i `batch_size=32` na skupovima za validaciju i testiranje ostvaruje iduću preciznost:

```
Epoch 1: valid accuracy = 67.930
Epoch 2: valid accuracy = 77.155
Epoch 3: valid accuracy = 78.254
Epoch 4: valid accuracy = 79.517
Epoch 5: valid accuracy = 80.011

Test accuracy = 79.985
```

Neovisno o tome koju ćeliju odaberete, pokrenite postupak učenja barem **5** puta s istim hiperparametrima ali različitim seedom. Pratite sve implementirane metrike i zapišite ih u datoteku.

Zadatak 4. Usporedba modela i pretraga hiperparametara (25% bodova)

Kao što vidimo, naše incijalne implementacije modela su dosta slične po preciznosti. Kako rezultati pokretanja modela za jedan skup hiperparametara mogu biti čista sreća ili nesreća, u ovom dijelu laboratorijske vježbe ćemo implementirati iscrpnu pretragu kroz varijante modela i njihove hiperparametre.

Usporedba RNN ćelija

Neovisno o tome koju RNN ćeliju ste odabrali u trećem zadatku, proširite vaš kod na način da vrsta RNN ćelije bude argument. Pokrenite vaš kod za preostale vrste RNN ćelija i zapišite rezultate. Je li neka ćelija očiti pobjednik? Je li neka ćelija očiti gubitnik?

Ponovite ovu usporedbu uz izmjenu hiperparametara povratnih neuronskih mreža. Idući hiperparametri povratnih neuronskih mreža su nam interesantni:

- hidden_size
- num_layers
- dropout: primjenjen **između** uzastopnih slojeva RNNa (funkcionira samo za 2+ slojeva)
- bidirectional: dimenzionalnost izlaza dvosmjerne rnn ćelije je **dvostruka**

Isprobajte **barem** 3 različite vrijednosti za svaki hiperparametar (osim bidirectional, koji ima samo dvije vrijednosti). Način na koji ćete kombinirati te vrijednosti je potpuno na vama (iscrpna rešetkasta pretraga je vremenski previše zahtjevna). Pokrenite svaku vrstu ćelije za svaku kombinaciju hiperparametara i zapišite rezultate (relevantne metrike). Nemojte se bojati raditi agresivne izmjene u vrijednostima hiperparametara (male izmjene vam neće dati puno informacija). Primjećujete li da neki hiperparametar bitno utječe na performanse ćelija? Koji?

Zapamtite / zapišite set hiperparametara koji vam daje najbolje rezultate. Za njega pokrenite učenje barem 5 puta s različitim seedovima i zapišite dobivene metrike.

Optimizacija hiperparametara

Probajte pokrenuti povratne neuronske mreže za najbolji set hiperparametara bez da koristite prednaučene vektorske reprezentacije. Probajte isto za vaš baseline model. Koji model više “pati” od gubitka prednaučenih reprezentacija?

Ulazne vektorske reprezentacije su jedan jako bitan hiperparametar, za koji u okviru laboratorijske vježbe imamo samo dvije vrijednosti – koristimo li ih ili ne. U analizi teksta su ulazne vektorske reprezentacije veoma velik dio uspješnosti algoritma. U ovom dijelu laboratorijske vježbe trebate odabrati **barem 5** od idućih hiperparametara te provjeriti kako modeli funkcioniraju za njihove izmjene. Ako hiperparametar utječe i na baseline model, kao i povratnu neuronsku mrežu, pokrenite eksperimente na oba modela. Za ćeliju povratne neuronske mreže odaberite onu koja ostvaruje (po vama) bolje rezultate na prošlom dijelu vježbe.

Za hiperparametre označene s nekim brojem zvjezdica (*), odaberite **samo jedan** od onih s istim brojem zvjezdica.

Hiperparametri:

- (*) Veličina vokabulara **V**
- (*) Minimalna frekvencija riječi **min_freq**
- (**) Stopa učenja
- (**) Veličina batcha
- Dropout
- Broj slojeva
- Dimenzionalnost skrivenih slojeva
- Optimizacijski algoritam (probajte nešto osim Adama)
- Funkcija nelinearnosti (u potpuno povezanim slojevima)
- Iznos na koji se podrezuju vrijednosti gradijenata
- Vrsta sažimanja (Baseline)
- Zamrzavanje ulaznih vektorskih reprezentacije (argument `freeze` funkcije `from_pretrained`)

Za svaki od odabranih hiperparametara isprobajte barem tri njegove različite vrijednosti (osim ako je binaran). Rezultate eksperimenata zapisujte. Pokrenite baseline i povratni model s najboljim hiperparametrima barem 5 puta i zapišite prosjek i devijaciju svih praćenih metrika. Čini li vam se neki parametar kao najutjecajniji za uspjeh? Nemojte se bojati raditi agresivne izmjene u vrijednostima hiperparametara jer će vas one lakše dovesti do zaključaka.

Upute oko eksperimenata i zapisivanja

Način i format zapisa rezultata je namjerno ostavljen otvoren. Cilj vježbe nije natjerati vas da radite u nekom okviru, već probajte ovo shvatiti kao istraživački projekt iz kojeg na kraju trebate nekome prezentirati svoja saznanja i potkrijepiti ih brojevima. Ti brojevi mogu biti zapis u tablici, kao dio slobodnog teksta s opisom eksperimenata ili vizualizirani. Rezultate ćete prezentirati verbalno, tako da odaberite format koji vam najviše paše za to.

Broj epoha je namjerno nedefiniran zbog različitih kapaciteta hardvera na vašim osobnim računalima. Skup podataka SST je odabran najviše jer je malen i čak ni izvođenje na CPU ne uzima previše vremena. Idejno bi se svaki eksperiment trebao pokrenuti na barem 5 epoha. Ukoliko vam to hardver vremenski ili prostorno ne dopušta, molimo vas da nam to javite.

Bonus zadatak: pozornost (max 20% bodova)

Dodatan zadatak biti će implementacija Bahdanau pozornosti za klasifikaciju slijeda. Konkretno, izbacujemo upit iz formulacije pozornosti, kao nelinearnost koristimo tangens hiperbolni a skrivena stanja (izlazi naše povratne neuronske mreže) će nam istovremeno služiti kao ključevi i vrijednosti.

Težine pozorosti računamo na idući način:

$$a^{(t)} = w_2 tanh(W_1 h^{(t)}) \quad \alpha = \text{softmax}(a)$$

A potom ih koristimo u težinskom sažimanju skrivenih stanja na idući način:

$$out_{attn} = \sum_{t}^{T} \alpha_t h^{(t)}$$

te izlaz algoritma pozornosti koristimo kao ulaz našem izlaznom sloju. Kao dimenziju skrivenog stanja algoritma pozornosti (izlazna dimenzija matrice [W_1]) koristite polovicu dimenzionalnosti skrivenog stanja vaše povratne neuronske mreže.

Neka u vašoj implementaciji korištenje algoritma pozornosti za povratnu neuronsku mrežu bude opcionalno (hiperparametar). Provjerite koliko pozornost doprinosi rezultatima svake neuronske ćelije. Pokrenite modele sa i bez algoritma pozornosti te zapišite prosjek i devijaciju svih praćenih metrika.

---

dlunizg
ivan.kreso@fer.hr