ДОДАТОК А

Програмний код досліджених алгоритмів

А.1 Згоркові нейронні мережі

```
    # keras.datasets.imdb is broken in 1.13 and 1.14, by np
1.16.3
    !pip install tf_nightly

    from __future__ import absolute_import, division,
print_function, unicode_literals

    import tensorflow as tf
    from tensorflow import keras

    import numpy as np

    print(tf.__version__)

    imdb = keras.datasets.imdb

    (train_data, train_labels), (test_data, test_labels) =
imdb.load_data(num_words=10000)

    print("Training entries: {}, labels:
{}".format(len(train_data), len(train_labels)))

    print(train_data[0])

    print(train_data[0])

    # A dictionary mapping words to an integer index
    word_index = imdb.get_word_index()

    # The first indices are reserved
    word_index = {k:(v+3) for k,v in word_index.items()}
    word_index["<PAD>"] = 0
    word_index["<START>"] = 1
    word_index["<UNK>"] = 2  # unknown
    word_index["<UNUSED>"] = 3

    reverse_word_index = dict([(value, key) for (key, value)
in word_index.items()])
```

```
    def decode_review(text):
        return ' '.join([reverse_word_index.get(i, '?') for
i in text])


    train_data                                              =
keras.preprocessing.sequence.pad_sequences(train_data,

value=word_index["<PAD>"],

padding='post',

maxlen=256)

    test_data                                               =
keras.preprocessing.sequence.pad_sequences(test_data,

value=word_index["<PAD>"],

padding='post',

maxlen=256)

    # input shape is the vocabulary count used for the movie
reviews (10,000 words)
    vocab_size = 10000

    model = keras.Sequential()
    model.add(keras.layers.Embedding(vocab_size, 16))
    model.add(keras.layers.GlobalAveragePooling1D())
    model.add(keras.layers.Dense(16,
activation=tf.nn.relu))
    model.add(keras.layers.Dense(1,
activation=tf.nn.sigmoid))

    model.summary()

    # input shape is the vocabulary count used for the movie
reviews (10,000 words)
    vocab_size = 10000

    model = keras.Sequential()
    model.add(keras.layers.Embedding(vocab_size, 16))
    model.add(keras.layers.GlobalAveragePooling1D())
```

```
    model.add(keras.layers.Dense(16,
activation=tf.nn.relu))
    model.add(keras.layers.Dense(1,
activation=tf.nn.sigmoid))

    model.summary()

    # input shape is the vocabulary count used for the movie
reviews (10,000 words)
    vocab_size = 10000

    model = keras.Sequential()
    model.add(keras.layers.Embedding(vocab_size, 16))
    model.add(keras.layers.GlobalAveragePooling1D())
    model.add(keras.layers.Dense(16,
activation=tf.nn.relu))
    model.add(keras.layers.Dense(1,
activation=tf.nn.sigmoid))

    model.summary()

    history = model.fit(partial_x_train,
                        partial_y_train,
                        epochs=40,
                        batch_size=512,
                        validation_data=(x_val, y_val),
                        verbose=1)

    results = model.evaluate(test_data, test_labels)

    print(results)


    history_dict = history.history
    history_dict.keys()

    import matplotlib.pyplot as plt

    acc = history_dict['acc']
    val_acc = history_dict['val_acc']
    loss = history_dict['loss']
    val_loss = history_dict['val_loss']

    epochs = range(1, len(acc) + 1)
```

```
    # "bo" is for "blue dot"
    plt.plot(epochs, loss, 'bo', label='Training loss')
    # b is for "solid blue line"
    plt.plot(epochs,   val_loss,   'b',   label='Validation
loss')
    plt.title('Training and validation loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    plt.show()

    plt.clf()   # clear figure

    plt.plot(epochs, acc, 'bo', label='Training acc')
    plt.plot(epochs, val_acc, 'b', label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.show()
```

Б.2 Рекурентні нейронні мережі

```
    import torch.nn as nn

    class SentimentLSTM(nn.Module):
        """
        The RNN model that will be used to perform Sentiment
analysis.
        """

        def    __init__(self,    vocab_size,    output_size,
embedding_dim, hidden_dim, n_layers, drop_prob=0.5):
            """
            Initialize the model by setting up the layers.
            """
            super().__init__()

            self.output_size = output_size
```

```
            self.n_layers = n_layers
            self.hidden_dim = hidden_dim

            # embedding and LSTM layers
            self.embedding    =    nn.Embedding(vocab_size,
embedding_dim)
            self.lstm = nn.LSTM(embedding_dim, hidden_dim,
n_layers,
                                dropout=drop_prob,
batch_first=True)

            # dropout layer
            self.dropout = nn.Dropout(0.3)

            # linear and sigmoid layers
            self.fc = nn.Linear(hidden_dim, output_size)
            self.sig = nn.Sigmoid()


        def forward(self, x, hidden):
            """
            Perform a forward pass of our model on some input
and hidden state.
            """
            batch_size = x.size(0)

            # embeddings and lstm_out
            embeds = self.embedding(x)
            lstm_out, hidden = self.lstm(embeds, hidden)

            # stack up lstm outputs
            lstm_out    =    lstm_out.contiguous().view(-1,
self.hidden_dim)

            # dropout and fully-connected layer
            out = self.dropout(lstm_out)
            out = self.fc(out)
            # sigmoid function
            sig_out = self.sig(out)

            # reshape to be batch_size first
            sig_out = sig_out.view(batch_size, -1)
            sig_out = sig_out[:, -1] # get last batch of
labels
```

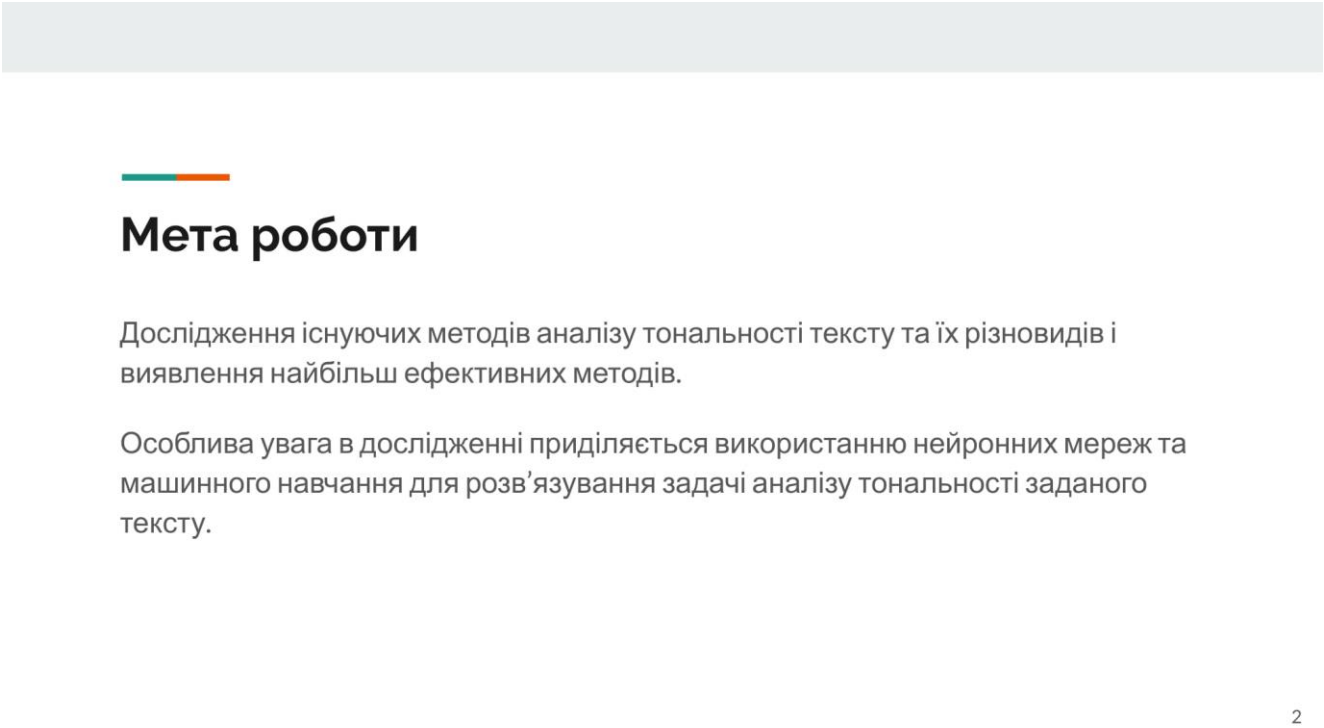
```
            # return last sigmoid output and hidden state
            return sig_out, hidden


        def init_hidden(self, batch_size):
            ''' Initializes hidden state '''
            # Create two new tensors with sizes n_layers x
batch_size x hidden_dim,
            # initialized to zero, for hidden state and cell
state of LSTM
            weight = next(self.parameters()).data

            if (train_on_gpu):
                hidden     =     (weight.new(self.n_layers,
batch_size, self.hidden_dim).zero_().cuda(),
                        weight.new(self.n_layers, batch_size,
self.hidden_dim).zero_().cuda())
            else:
                hidden     =     (weight.new(self.n_layers,
batch_size, self.hidden_dim).zero_(),
                            weight.new(self.n_layers,
batch_size, self.hidden_dim).zero_())

            return hidden

    # loss and optimization functions
    lr=0.001

    criterion = nn.BCELoss()
    optimizer = torch.optim.Adam(net.parameters(), lr=lr)


    # training params

    epochs = 4 # 3-4 is approx where I noticed the validation
loss stop decreasing

    counter = 0
    print_every = 100
    clip=5 # gradient clipping

    # move model to GPU, if available
    if(train_on_gpu):
        net.cuda()
```

```
    net.train()
    # train for some number of epochs
    for e in range(epochs):
        # initialize hidden state
        h = net.init_hidden(batch_size)

        # batch loop
        for inputs, labels in train_loader:
            counter += 1

            if(train_on_gpu):
                inputs,     labels     =     inputs.cuda(),
labels.cuda()

            # Creating new variables for the hidden state,
otherwise
            #  we'd backprop through the entire training
history
            h = tuple([each.data for each in h])

            # zero accumulated gradients
            net.zero_grad()

            # get the output from the model
            inputs = inputs.type(torch.LongTensor)
            output, h = net(inputs, h)

            # calculate the loss and perform backprop
            loss       =       criterion(output.squeeze(),
labels.float())
            loss.backward()
            # `clip_grad_norm` helps prevent the exploding
gradient problem in RNNs / LSTMs.
            nn.utils.clip_grad_norm_(net.parameters(),
clip)
            optimizer.step()

            # loss stats
            if counter % print_every == 0:
                # Get validation loss
                val_h = net.init_hidden(batch_size)
                val_losses = []
                net.eval()
                for inputs, labels in valid_loader:
```

```
                    # Creating new variables for the hidden
state, otherwise
                    #  we'd  backprop  through  the  entire
training history
                    val_h  =  tuple([each.data  for  each  in
val_h])

                    if(train_on_gpu):
                        inputs,  labels  =  inputs.cuda(),
labels.cuda()

                    inputs = inputs.type(torch.LongTensor)
                    output, val_h = net(inputs, val_h)
                    val_loss  =  criterion(output.squeeze(),
labels.float())

                    val_losses.append(val_loss.item())

                net.train()
                print("Epoch:         {}/{}...".format(e+1,
epochs),
                      "Step: {}...".format(counter),
                      "Loss:
{:.6f}...".format(loss.item()),
                      "Val                              Loss:
{:.6f}".format(np.mean(val_losses)))
```

ДОДАТОК Б

Слайди презентації



Аналіз тональності (sentiment analysis)

Клас методів аналізу контенту в комп'ютерній лінгвістиці, що призначений для автоматизованого виявлення в текстах емоційно забарвленої лексики і емоційної оцінки по відношенню до об'єктів, мова про які йде в тексті.

3

Проблема класифікації

1. класифікація речення як суб'єктивна або об'єктивна -- класифікація суб'єктності;
2. класифікація речення як вираження позитивного або негативного -- класифікація полярності.

4

Області застосування аналізу тональностей

- моніторинг соціальних медіа
- моніторинг брендів
- аналіз ринку
- класифікація відгуків
- класифікація важливості звернень у службу підтримки
- аналітика робочого процесу
- etc.

5

Існуючі методи

1. системи на основі правил
2. автоматичні системи, які спираються на методи машинного навчання
3. гібридні системи

6

Досліджувані методи

1. З використанням згорткових нейронних мереж
2. З використанням рекурентних нейронних мереж
3. З використанням transfer learning

7

Згоркові нейронні мережі

Перехід від конкретних особливостей вхідних даних до більш абстрактних деталей, і далі до ще більш абстрактних деталей, доходячи до виділення понять високого рівня.

Є регуляризованною версією багатошарового персептрона, який розроблено таким чином, щоб потрібно було проводити мінімальну попередню обробку

8

Етапи і результати дослідження

1. підраховується кількість використань кожного слова у датасеті;
2. будується словник використаних слів;
3. кожний відгук приводиться до довжини в 256 слів;
4. кожний відгук конвертується в тензор;
5. будується модель нейронної мережі;
6. проводиться навчання нейронної мережі;
7. перевіряється результат роботи.

Точність передбачень

0,8699

9

Рекурентні нейронні мережі

Міжвузлові з'єднання утворюють направлену у часі послідовність (орієнтований граф)

Можуть використовувати свою внутрішню пам'ять для обробки послідовностей довільної довжини

Варіанти архітектури:

- з довгою короткочасною пам'яттю (LSTM)
- з керованим рекурентним блоком (GRU)

10

Етапи і результат дослідження

1. обробка даних;
2. токенізація;
3. аналіз довжини відгуків, приведення усіх відгуків до спільної середньої довжини;
4. визначення мережевої архітектури LSTM;
5. побудування класу моделі;
6. навчання мережі;
7. тестування мережі.

Точність передбачень:

0,884

11

Transfer learning

Модель, що навчається на одному завданні, переорієнтована на друге відповідне завдання

Відомі попередньо треновані моделі:

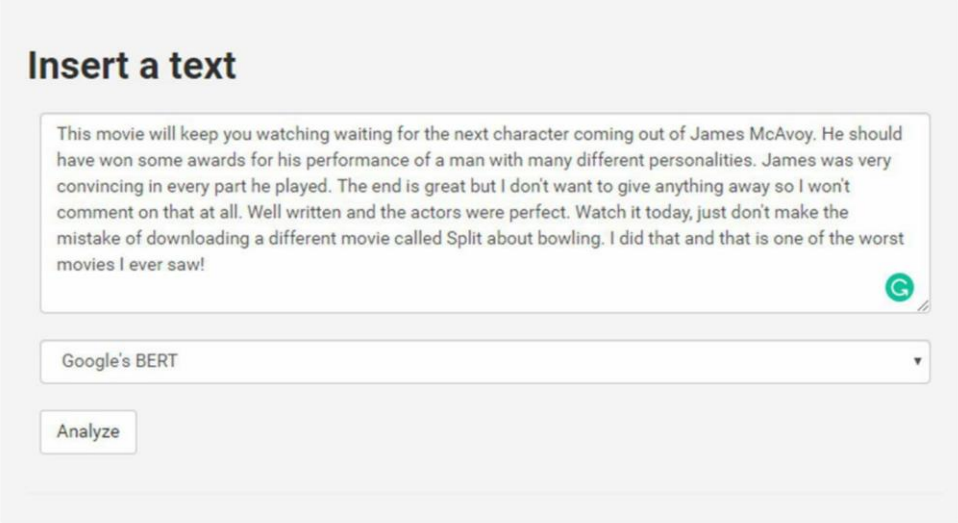
- ULMFiT
- Transformer
- Google's BERT
- ELMo

Точність передбачень:

0,856

12

Додаток для тестування



Підсумкові результати

Датасет	Згорткова мережа	LSTM-мережа	BERT
Відгуки IMDB	0,86	0,87	0,85
Twitter US Airline Sentiment	0,67	0,701	0,83
Paper Reviews	0,76	0,79	0,84
Amazon Reviews	0,78	0,81	0,84

Висновки

Якщо необхідно проводити аналіз текстів конкретної предметної області (наприклад, аналіз звернень до служби підтримки), то більш ефективним є використання нейронної мережі, натренованої на тренувальних даних тієї ж самої предметної області.

Скщо система повинна визначати тональність довільних текстів, для яких не можна виділити спільне джерело чи тематику, має сенс використовувати підхід Transfer Learning, тобто використовувати попередньо треновані моделі.

15

Дякую за увагу

16

ДОДАТОК В

Апробація результатів роботи

ВИКОРИСТАННЯ МЕТОДІВ TRANSFER LEARNING ДЛЯ АНАЛІЗУ ТОНАЛЬНОСТЕЙ ТЕКСТУ

БАБАСКІНА І.О.
iryna.babaskina@nure.ua
студент кафедри Програмної інженерії
Харківській національний університет радіоелектроніки
м.Харків, Україна

Аналіз тональності текстів – це клас методів контент-аналізу, призначений для автоматичного виявлення в тексті емоційно забарвленої лексики, а також думок (емоційних оцінок) автора з приводу об'єктів, в яких йде мова в тексті[1]. Існує декілька типів систем, що розв’язують задачі визначення тональності тексту:

1. Системи на основі правил, які виконують аналіз настроїв на основі набору ручних правил.
2. Автоматичні системи, які спираються на методи машинного навчання і замість правил використовують відомі дані.
3. Гібридні системи, які поєднують обидва підходи, правила і автоматичні.

Найбільш цікавими для дослідження є автоматичні системи, які спираються на методи машинного навчання. Зараз існує достатньо датасетів, спираючись на які, можна навчити нейронну мережу визначати тональність заданого тексту. Основна складність полягає в тому, що для навчання нейронноїх мережі потрібно багато обчислювальних ресурсів.

Цю проблему вирішує підхід трансферного навчання (transfer learning). Трансферне навчання – це поліпшення навчання новій задаія через передачу знань з пов'язаної задачі, яка вже була вивчена[2]. Ідея трансферного навчання полягає в тому, що якщо модель навчена на великому і загальному наборі даних, то ця модель буде ефективно служити загальною моделлю візуального світу. Потім є можливість скористатися цими вивченими картами властивостей без необхідності починати з нуля навчання великої моделі на великому наборі даних[2]. Хоча більшість алгоритмів машинного навчання розроблені для вирішення окремих завдань, задача аналізу тональностей тексту відноситься до тих, у яких використання трансферного навчання може надати значні переваги, оскільки задача аналізу тональності тексту є загальною і більше залежить від мови текстів, ніж від конкретної тематики текстів.

Для дослідження пеерваг трансферного навчання для задачі визначення тональності тексту використовуються два типи нейронних мереж: згорткова нейронна мережа та рекурентна нейронна мережа (у варіанті архітектури коротка

довгочасна пам’ять, LSTM). Навчання мережі проводиться на корпусі відгуків до фільмів з сайту IMDB.

Для підготовки даних для опрацювання нейронною мережею прозводиться процедура токенізаціїї – перетворення відгуків в word embeddings. Це, по суті, форма представлення слова, яка перекриває людське розуміння мови та розуміння мови. Word embeddings є розподіленими поданнями тексту в n-мірному просторі для подальшої обробки нейронними мережами. Word embeddings представлені у вигляді багатомірних векторів. Для побудування цих векторів кожному слові у корпусі даних відводиться цифрове значення в залежності від вживаності цього слова (так, для найбільш вживаного слова “the” це значення буде 1, і т.д. до найменш вживаного.

Для створення класифікатора на основі згорткової нейронної мережі всі верстви проходять процес стека, або накладення:

а) перший Embedding шар приймає перекладені в цілі числа слова і шукає відповідний вектор для кожної пари слово/число. Модель навчається на цих векторах. Вектори збільшують розмір одержуваного масиву на 1, в результаті чого ми отримуємо вимірювання: (batch, sequence, embedding)[3];

б) наступний шар GlobalAveragePooling1D повертає отриманий вектор заданої довжини для кожного прикладу, усереднюючи розмір ряду. Це дозволить моделі легко приймати дані різної довжини;

в) цей вектор пропускається через повнозв'язний Dense шар з 16 прихованими блоками;

г) останній шар також є повнозв'язним, але з усього одним вихідним вузлом. За допомогою функції активації sigmoid (сигмоид) отримується число з плаваючою комою між 0 і 1, яке буде показувати ймовірність або впевненість моделі.

Вищеописана модель має 2 проміжних або приховані прошарки, між входом і виходом даних. Кількість виходів (блоків, нодів або нейронів) є розміром репрезентативного простору шару. Іншими словами, кількість свободи, яка дозволена мережі під час навчання[4].

Якщо модель має більше прихованих блоків, і/або більше шарів, то тоді нейросеть може навчитися більш складним уявленням. Однак в цьому випадку це буде дорожче з точки зору обчислювальних ресурсів і може призвести до навчання небажаних патернів – патернів, які покращують показники на тренувальних даних, але не на перевірочних. Це називається перенавчанням. Експериментально було виявлено, що для уникнення перенавчання нейронну мережу варто навчати на двадцяти епохах. Процент точності – 87%.

Для LSTM-моделі спочатку, визначаються наступні гіперпараметри:

а) lstm_size: Кількість одиниць у прихованих шарах у клітинах LSTM. Зазвичай більше значення краще с точки зору ефективністі. Загальні значення 128, 256, 512 і т.д., але використовується середня довжина відгуку;

б) lstm_layers: кількість шарів LSTM в мережі. Я б почав з 1, а потім додавав більше, якщо я недостатньо підібраний;

в) batch_size: кількість відгуків для передачі мережі в одному навчальному пропуску. Як правило, це має бути встановлено настільки високим, як можна перейти без пам'яті;

г) learning_rate: швидкість навчання.

Обрано наступні параметри:

lstm_size = 256

lstm_layers = 2

batch_size = 1000

learning_rate = 0.01

Архітектура побудованної нейронної мережі складається з наступних шарів:

а) embedding шар, що перетворює слова (цілі числа) у вектори певного розміру;

б) шар LSTM: визначається вимірами схованого стану і кількістю шарів;

в) повністю підключений шар: що відображає вихід LSTM-шару на бажаний розмір виводу;

г) рівень активації сигмоїдів: перетворює всі вихідні значення у значення від 0 до 1;

ґ) вихідні дані: Сигмоподібний вихід з останнього тимчасового кроку розглядається як кінцевий вихід цієї мережі.

Після 15 епохи починається перенавчання моделі. Результати точності: 88,4%.

Таким чином, маємо дві попередньо натреновані моделі нейронних мереж. Далі ці моделі можна використовувати для розв’язання задач визначення тональності тексту. Перевірка моделей проводиться на датасетах Amazon Reviews for Sentiment Analysis (відгуки клієнтів Amazon) та Twitter US Airline Sentiment (набір твітів про авіакомпанії США). Отримана в результаті точність менша, ніж точність для текстів з того ж самого датасету, що і тренувальні дані.

Датасет	Згорткова мережа	LSTM-мережа
Відгуки IMDB	0,86	0,87
Twitter US Airline Sentiment	0,77	0,791
Amazon Reviews	0,78	0,81

На датасеті, тематика і розмір текстів якого ближче до того датасету, на якому проводилося навчання моделей, попередньо навчені моделі показують кращий результат, ніж на датасеті з меншим середнім обсягом даних (база твітів). Це є очікуваним результатом, і результати класифікації для інших датасетів є цілком прийнятними.

Таким чином, задача визначення тональності тексту є однією з тих, де доцільно використовувати підхід трансферного навчання. Використання попередньо тренованих моделей зменшує накладні витрати на навчання систем, а також вирішує проблему тренувальних даних для випадків, коли реальні дані є досить варіативними і не можуть буди однозначно віднесені до певного набору. Точність у випадку використання трансферного навчання менше, ніж при

виявленні тональності текстів з того ж корпусу, що й навчальні дані, але все ще є прийнятною для даної задачі.

Використана література:

[1] Gerardus Blokdyk, Sentiment Analysis a Complete Guide // 5starcooks – p. 32-37, 2018

[2] Ian Goodfellow, Yoshua Bengio, Deep Learning (Adaptive Computation and Machine Learning series) // The MIT Press – p. 621-638, 2016.

[3] Bianchi, F.M., Maiorino, E., Recurrent Neural Networks for Short-Term Load Forecasting // SpringerBriefs – p. 45-48, 2017

[4] Frank Millstein, Deep Learning: 2 Manuscripts - Deep Learning With Keras And Convolutional Neural Networks In Python // Paperback – p.117-129, 2018