МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ
УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

INTELLECTUAL DATA ANALYSIS

Лабораторний практикум №3.

Виконав
студент гр. ФБ-51мп
Цибулено-Сігов І. М.
Перевірив

Київ 2025

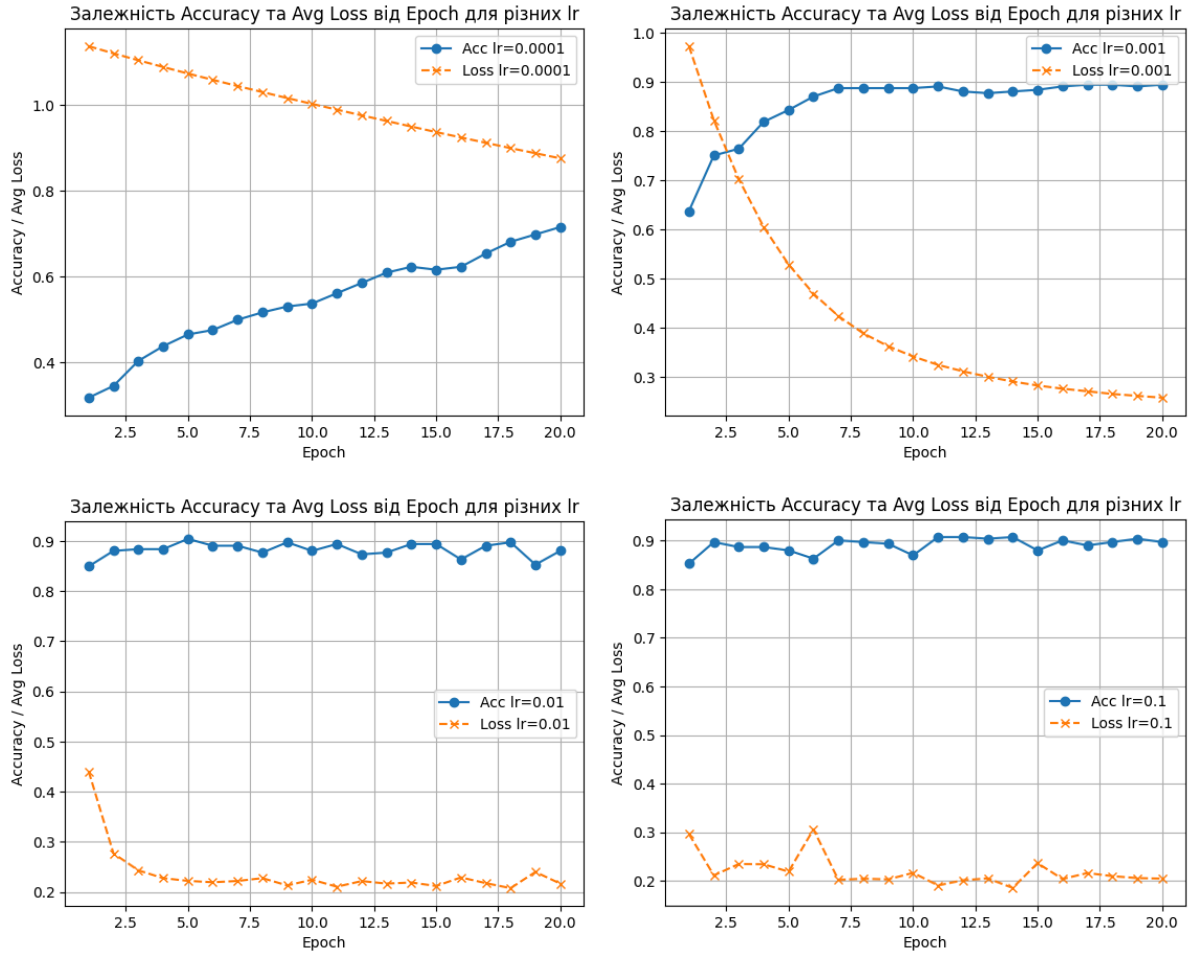
1. Повнозв'язані нейронні мережі

Візьміть дані, з якими ви працювали в лабораторній №1. Побудуйте повнозв’язану нейронну мережу прямого поширення для задачі класифікації. Навчіть її на тренувальній вибірці, протестуйте на тестовій. Порівняйте результати з алгоритмами з Lab 1.

Трохи погравшись із кількістю шарів та нейронів зупинився на такій кількості:

```
FFNN(
  (model): Sequential(
    (0): Linear(in_features=5, out_features=16, bias=True)
    (1): ReLU()
    (2): Linear(in_features=16, out_features=3, bias=True)
  )
)
```

Також досліджу як себе веде модель при різних швидкостях навчання



Щоб зберегти стабільність залишаю lr = 0.001 а кількість епох - 20

За таких параметрів отримую результат:

```
Final Test Accuracy: 0.8938356164383562
```

В першій лабораторній найкраще справився алгоритм Decision tree

```
Точність моделі на тестовій вибірці: 0.9144
Точність моделі на тренувальній вибірці: 0.8987
```

Якщо порівнювати ці моделі, то вони однаково добре прогнозують класи (різниця на рівні похибки)

2. Згорткові нейронні мережі
а) Побудуйте просту згорткову нейронну мережу
(2–3 convolutional шари + fully connected). Навчіть її на обраному вами датасеті.
На цьому етапі у мене закінчилисчя кредити для онлайн розрахунків - пересідаю на локалхост.


```
Epoch 30/30
Train loss: 0.1537 | Train acc: 0.9448 | Val loss: 3.0452 | Val acc: 0.5497
```

З графіку явно видно перенавчання моделі. Також порівняв свій результат із результатом ентузіастів із kaggle - у них в середньому виходить 60% на 30 епосі.

Через те, що модель тренується довго - не вистачить часу проаналізувати як саме впливає кожен параметр на модель, одразу внесу кілька змін.

Було збільшено кількість згорткових шарів з двох до трьох, що дозволяє моделі більш глибоко аналізувати локальні патерни обличчя та покращує здатність розпізнавати складні емоційні вирази. Кожен згортковий шар тепер супроводжується нормалізацією батчів (Batch Normalization), що стабілізує процес навчання та зменшує ризик перенавчання. Крім того, перед повнозв'язковими шарами доданий Dropout з ймовірністю 0.4, що додатково знижує ймовірність переобучення на тренувальних даних.

Також було впроваджено значні зміни в обробці даних. Для тренувального набору застосовані методи Data Augmentation: випадкові горизонтальні віддзеркалення, обертання, а також легка зміна яскравості та контрасту. Це допомагає моделі стати більш стійкою до варіацій у зображеннях облич і покращує узагальнення на валідаційних даних. Тестовий набір залишився без змін, забезпечуючи адекватну оцінку продуктивності моделі.

Крім того, було додано автоматичне регулювання швидкості навчання через ReduceLROnPlateau, що дозволяє адаптувати learning rate при сповільненні зменшення функції втрат. Для уникнення надмірного навчання використаний механізм ранньої

зупинки (Early Stopping), який припиняє тренування, якщо точність на валідації не покращується протягом певної кількості епох.


```
Epoch 28/30

Train Loss: 1.0354 | Train Acc: 0.6061 | Val Loss: 1.1145 | Val Acc: 0.5965

Epoch 29/30

Train Loss: 1.0236 | Train Acc: 0.6152 | Val Loss: 1.0388 | Val Acc: 0.6119

Epoch 30/30

Train Loss: 1.0119 | Train Acc: 0.6158 | Val Loss: 1.0436 | Val Acc: 0.6094
```

Тепер модель виглядає краще. з графіку видно, що якщо додати більше епох, можна отримати і кращк точність, але часу на це піде чимало.
61% точності для цього датасету вже доволі непоганий результат. На цьому зупинюсь.

б) Використайте попередньо натреновану архітектуру (наприклад, ResNet, VGG, MobileNet). Замініть вихідний класифікатор на новий під ваші класи. Проведіть донавчання моделі на вашому датасеті. Порівняйте результати (точність, швидкість збіжності, кількость даних).

```
Epoch 1/15

Train Loss: 1.1484 | Train Acc: 0.5638 | Val Loss: 0.9929 | Val Acc: 0.6303

Epoch 2/15

Train Loss: 0.9163 | Train Acc: 0.6578 | Val Loss: 0.9608 | Val Acc: 0.6457
```

Донавчання відбувається дуже довго ~ 15 хв на 1 епоху.

Перша ж епоха одразу переплюнула на 2% по точності на тестових даних 30 епоху моделі, що навчалась з нуля.

```
Epoch 6/15

Train Loss: 0.5552 | Train Acc: 0.8001 | Val Loss: 0.9624 | Val Acc: 0.6769

Epoch 7/15

Train Loss: 0.4829 | Train Acc: 0.8267 | Val Loss: 1.0383 | Val Acc: 0.6718

Epoch 8/15

Train Loss: 0.3278 | Train Acc: 0.8880 | Val Loss: 1.0059 | Val Acc: 0.6955

Epoch 9/15

Train Loss: 0.2553 | Train Acc: 0.9171 | Val Loss: 1.0558 | Val Acc: 0.6877

Epoch 10/15

Train Loss: 0.2163 | Train Acc: 0.9296 | Val Loss: 1.1337 | Val Acc: 0.6849
```

пік точності досягнуто на 8 епосі, далі модель збігається.

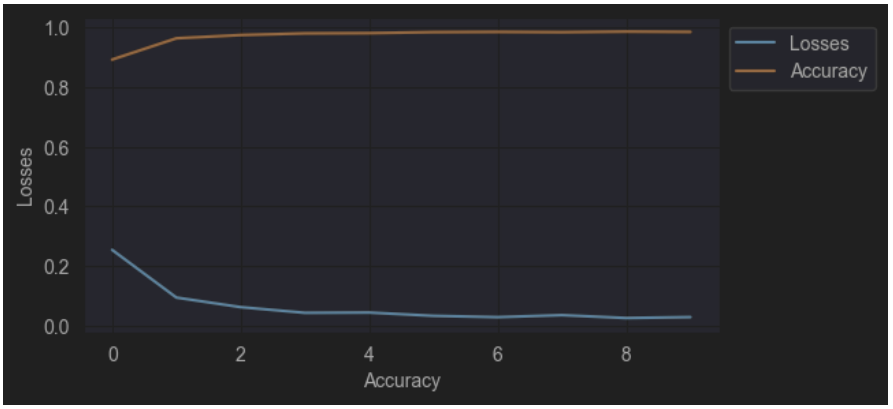
```
Epoch 14/15

Train Loss: 0.0940 | Train Acc: 0.9727 | Val Loss: 1.2231 | Val Acc: 0.6913

Epoch 15/15

Train Loss: 0.0853 | Train Acc: 0.9747 | Val Loss: 1.2544 | Val Acc: 0.6948
```

Модель вийшла перенавченою, що видно з різниці точностей на тренувальних та тестових даних.

3. Вирішіть задачу класифікації текстів (використайте той же датасет, з яким ви працювали в лабораторній № 2) двома способами:
а) Побудуйте модель з вбудованим Embedding шаром (ініціалізованим випадковими вагами). Використайте RNN / LSTM / GRU для класифікації. Навчіть модель на вашому датасеті.
б) Завантажте готові embeddings (наприклад, GloVe). Ініціалізуйте Embedding шар цими вагами. Проведіть навчання.
Порівняйте якість класифікації у (а) та (б). Чи покращилися метрики при використанні pretrained embeddings? Наскільки швидше/стабільніше відбулося навчання?
Я не розібрався як використати саме Pytorch для цієї задачі, тому використовуватиму іншу бібліотеку – keras.

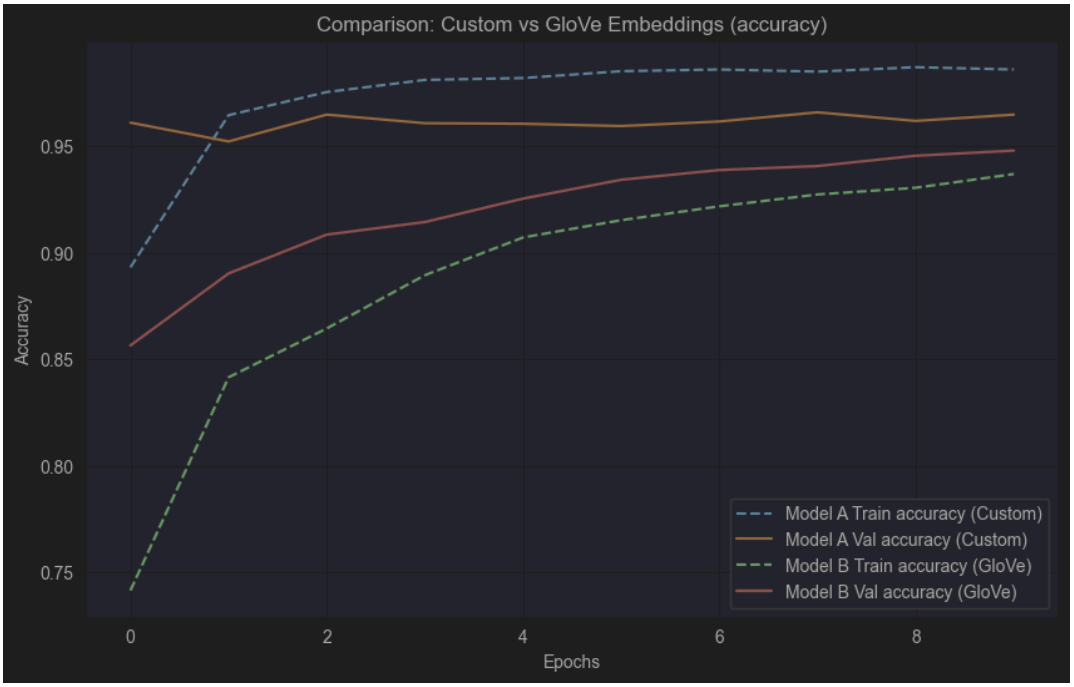
Нейромережі не розуміють текст, вони розуміють числа. Тому перетворюю очищені тексти на послідовності індексів. (from keras.preprocessing.text import Tokenizer)
Вибрав LSTM для класифікації.

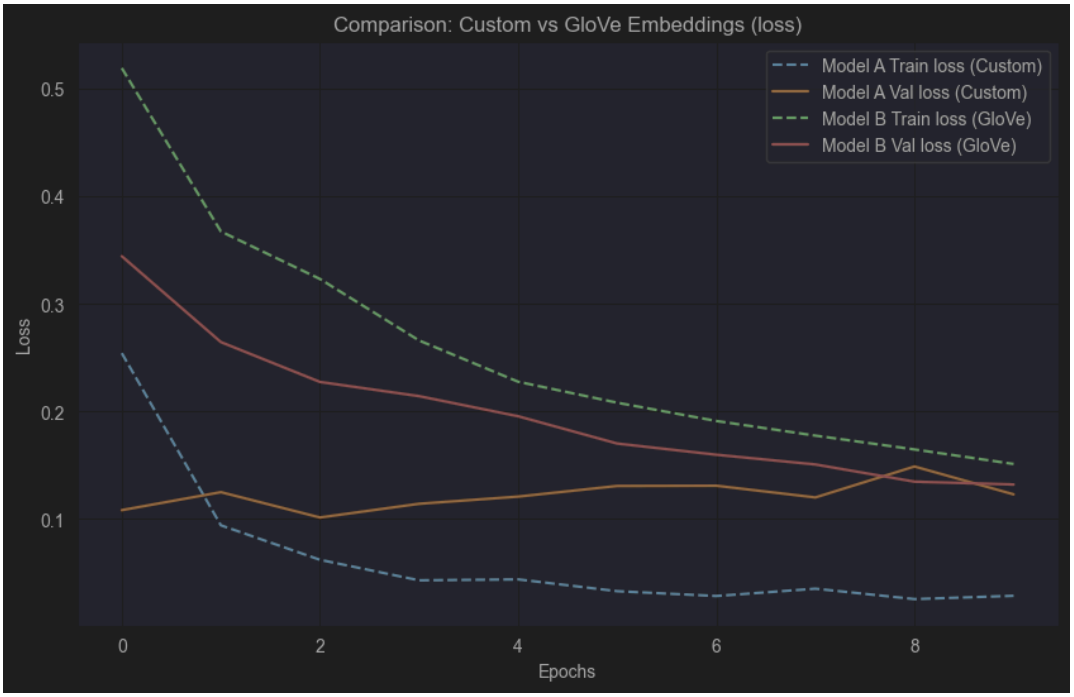
```
Epoch 8/10
233/233 ━━━━━━━━━━━━━━━━━━━━ 9s 38ms/step - accuracy: 0.9850 - loss: 0.0352 - val_accuracy: 0.9659 - val_loss: 0.1201
Epoch 9/10
233/233 ━━━━━━━━━━━━━━━━━━━━ 9s 38ms/step - accuracy: 0.9871 - loss: 0.0255 - val_accuracy: 0.9619 - val_loss: 0.1489
Epoch 10/10
233/233 ━━━━━━━━━━━━━━━━━━━━ 9s 39ms/step - accuracy: 0.9860 - loss: 0.0286 - val_accuracy: 0.9649 - val_loss: 0.1230
```



Класифікація пройшла супер. Точність збігається із результатом Random Forest.

Для b) завантажую GloVe: http://nlp.stanford.edu/data/glove.6B.zip
використаю набір векторів glove.6B.100d.txt





У першому випадку я використав Embedding-шар з випадковою ініціалізацією, дозволивши моделі самостійно формувати векторні представлення слів у процесі навчання. У другому випадку я ініціалізував вхідний шар готовими вагами GloVe, попередньо навченими на великих масивах текстів. Із графіків навчання можна помітити суттєву різницю: модель з власними ембедінгами швидко досягла високої точності (близько 97%), але продемонструвала перенавчання (Loss на валідації почав зростати, а на навчання стрімко наближається до 0). Натомість модель з GloVe навчалася значно стабільніше, показуючи мінімальний розрив між тренувальними та валідаційними даними, хоча й досягла трохи нижчої пікової точності (близько 95%).

Такий результат зумовлений лексичними особливостями датасету. Фішингові листи насичені специфічним сленгом, навмисними помилками та технічними артефактами (наприклад слова типу "urgent!!!"), які відсутні у стандартному літературному словнику GloVe. Тому модель, що навчалася «з нуля», змогла краще адаптуватися до цих аномалій і виділити характерні ознаки спаму, тоді як універсальні ваги GloVe виявилися занадто загальними для цієї вузькоспеціалізованої задачі, хоча й забезпечили кращу узагальнюючу здатність мережі.