Τεχνητή Νοημοσύνη

Ai Assignment 2

Βασίλης Τσιάτσιος p3160178

- **Μέρος Α**

- Εξωτερικές Βιβλιοθήκες Μέρους Α.....
  Preprocess.py.....
  Models.py.....
  Train_dev_test.py.....
  Learning_curves.py.....
  Evaluate.py.....

- **Μέρος Β**

- Εξωτερικές Βιβλιοθήκες Μέρους Β.....
  dataset_rnn.py.....
  glove_loader.py.....
  rnn_model.py.....
  train_rnn.py.....
  evaluate_rnn.py.....

- **Μέρος Γ**

- Εξωτερικές Βιβλιοθήκες Μέρους Γ.....
  cnn_model.py.....
  train_cnn.py.....
  evaluate_cnn.py.....

- Φωτογραφίες/Πινακάκια

Meros A

Για την επίτευξη του πρώτου κομματίου δημιουργήθηκαν 5 αρχεια

Το preprocess.py,Models.py,train_dev_test.py,learning_curves.py,Evaluate.py

Πριν προχωρήσω στο τι είναι το καθένα για αυτό το κομμάτι χρησιμοποιήθηκαν

Εξωτερικές Βιβλιοθήκες

NumPy για Πίνακες, μαθηματικές πράξεις (log2, mean), random permutation
schikit-learn για Αλγόριθμους ML (BernoulliNB, RandomForest, LogisticRegression), metrics (precision_recall_fscore_support), train_test_split
matplotlib Γραφήματα καμπυλών μάθησης
joblib Αποθήκευση/φόρτωση vocab.pkl και models.pkl
Στο Μερος Α και πιο συγκεκριμένα στο αρχείο preprocess.py,έπρεπε να φορτώσουμε τα μεταδεδομένα απο τον τοπικό φάκελο imdb/train/ και imdb/test/. Κάθε φάκελος περιέχει δύο υποφακέλους:
pos/ → θετικές κριτικές (label = 1)
neg/ → αρνητικές κριτικές (label = 0)
Η συνάρτηση load_dataset(split) διαβάζει όλα τα .txt αρχεία, τους αναθέτει ετικέτες, και τα ανακατεύει τυχαία.
Έπειτα χρησιμοποιούμε την συνάρτηση tokenize(text) για να μετατρέψει το κείμενο σε πεζά και αποθηκεύει όλους τους χαρακτήρες με regex re.findall(r"[a-zA-Z]+", text)
και επιστρέφει λίστα απο tokens(λέξεις)Στην σύνεχεια η συνάρτηση compute_document_frequency(token_lists) υπολογίζει πόσα έγγραφα περιέχουν κάθε λέξη (όχι πόσες φορές εμφανίζεται συνολικά).

Χρησιμοποιεί set(tokens) για κάθε κείμενο, ώστε μια λέξη να μετράει μία φορά ανά έγγραφο.

```
def compute_document_frequency(token_lists):
df = Counter()
  for toks in token_lists:
      df.update(set(toks))
   return df
```

set(toks) → κρατάει κάθε λέξη μία φορά ανά έγγραφο

Counter μετράει σε πόσα έγγραφα εμφανίζεται κάθε λέξη
Δεν μετράει πόσες φορές εμφανίζεται μέσα στο ίδιο έγγραφο

Συνεχίζοντας , γίνεται αφαίρεση συχνών και σπάνιων λέξεων με σκοπό να καθαρίσει το λεξιλόγιο
πριν τον Υπολογισμό του Information Gain. Αφαιροντας λεξεις που δεν εχουν πληροφορία συναισθήματος καθώς και τις πιο σπάνιες λέξεις θα βγει ενα καλύτερο αποτέλεσμα.Γίνεται με την συνάρτηση mpute_information_gain(df, token_lists, labels).Αυτό γινέται με ταξινόμηση κατά φθίνουσα σείρα και επιστρέφει το σύνολο των λέξεων προς αφαίρεση
Τώρα είμαστε έτοιμη για Information Gain οπού υπολογίζει το πληροφοριακό κέρδος κάθε λέξης.
Υπολογίζουμε την εντροπία της κατανομής H(Y),δημιουργουμε dictionary type word → [indices εγγράφων που την περιέχουν] IG(word) = H(Y) - P(word)·H(Y|word) - P(¬word)·H(Y|¬word)
οπού H(Y) = εντροπία της κατανομής των κλάσεων (θετικές/αρνητικές)
H(Y|word) = εντροπία όταν η λέξη υπάρχει
H(Y|¬word) = εντροπία όταν η λέξη δεν υπάρχει
Υψηλό IG = η λέξη διαχωρίζει καλά τις κλάσεις (π.χ. "excellent" → θετικό, "terrible" → αρνητικό)
Έπειτα παμε για κατασκευή λεξιλογίου
Η συνάρτηση build_vocabulary(texts, labels, n=200, k=5, m=3000):

Κάνει tokenization
Υπολογίζει DF
Αφαιρεί τις n πιο συχνές και k πιο σπάνιες
Υπολογίζει IG για τις υπόλοιπες
Επιλέγει τις m=3000 λέξεις με το υψηλότερο IG
Το αποτέλεσμα είναι ένα dictionary: vocab[word] = index

Τέλος για το αρχείο αυτό μέσω του vectorize function
Δημιουργεί πίνακα (N κείμενα) × (3000 λέξεις)
X[i, j] = 1 αν το κείμενο i περιέχει τη λέξη j αλλιώς
X[i, j] = 0 αν δεν.
set(toks) → κάθε λέξη μετράει μία φορά (binary, όχι count)

Στο αρχείο models.py

οπού ορίζονται τα modela που χρησιμοποιούμε
Έχουμε 3 modela
BernoulliNB: Naive Bayes για binary features.
Υποθέτει ότι κάθε feature (λέξη) είναι ανεξάρτητο.
Default παράμετροι

RandomForest: Τυχαίο Δάσος με 200 δέντρα | n_estimators=200, max_features="sqrt"
Κάθε δέντρο εκπαιδεύεται σε random subset δεδομένων και features.

LogisticRegression: Γραμμικό μοντέλο που μαθαίνει βάρη για κάθε λέξη.
liblinear solver είναι γρήγορος για sparse data.
solver="liblinear", max_iter=3000

Μετά πάμε στην εκπαίδευση στο αρχείο train_dev_test.py

Διαδικασία:

Φόρτωση train set (25.000 κριτικές) // texts, labels = load_dataset("train")
Split σε 80% train / 20% dev (stratified)
stratify=labels → διατηρεί την αναλογία κλάσεων (50-50)
Κατασκευή vocabulary από το train set
Vectorization train και dev
Εκπαίδευση κάθε μοντέλου
Αξιολόγηση στο dev set (Precision, Recall, F1)
Αποθήκευση vocab.pkl και models.pkl

Το αρχείο learning_curves.py

που είναι οι καμπύλες μάθησης ,εκπαιδεύει με διαφορετικά μεγέθη train set,Μετράει performance σε train subset και ολόκληρο dev set,Δείχνει πώς βελτιώνεται το μοντέλο με περισσότερα δεδομένα και παράγει γραφήματα για precision, recall, F1

Τέλος για το Μέρος Α το evaluate.py

Φορτώνει το test set (25k κριτικές)
Χρησιμοποιεί το ίδιο vocabulary από το training
Vectorize με τον ίδιο τρόπο και κρατάει τις μετρικές

Υπολογισμός metrics:
Per-class: Precision, Recall, F1 για neg και pos
Micro average: συνολικό F1
Macro average: μέσος όρος F1 των δύο κλάσεων

Υπερπαραμετροί Α
n =200, Αφαίρεση των 200 πιο συχνών λέξεων
k =5,Αφαίρεση των 5 πιο σπάνιων λέξεων
M=3000 ,Μέγεθος λεξιλογίου (top IG words)
min_df=5, Ελάχιστο document frequency
RandomForest, n_estimators =200
max_features = "sqrt"
LogisticRegression ,solver= "liblinear"
max_iter=3000
C 1.0 (default)
BernoulliNB defaults

Train/Dev Split

Μέρος Β

Για την επίτευξη του δεύτερου κομματιού δημιουργήθηκαν 5 αρχεία:

dataset_rnn.py
glove_loader.py
rnn_model.py
train_rnn.py
evaluate_rnn.py

Εξωτερικές Βιβλιοθήκες

PyTorch Νευρωνικά δίκτυα, GRU layers, embeddings, training loop, optimizer
torchtext Φόρτωση του IMDB dataset

NumPy Αποθήκευση embedding matrix, μαθηματικές πράξεις
matplotlib Γραφήματα loss curves
joblib Αποθήκευση/φόρτωση vocab.pkl
scikit-learn classification_report για metrics

Αρχείο dataset_rnn.py

είναι υπεύθυνο για την προεπεξεργασία των δεδομένων και την προετοιμασία τους για το RNN.
Μέσω της tokenize(text)
Μετατρέπει το κείμενο σε πεζά
Αφαιρεί όλους τους ειδικούς χαρακτήρες (κρατάει μόνο γράμματα, αριθμούς, κενά)
Σπάει το κείμενο σε λέξεις με split().Έπειτα κατασκευάζει Vocabulary που μετράει πόσες φορές εμφανίζεται κάθε λέξη και κρατάει συχνότητα ≥ 2.
Προστέθηκαν ειδικά tokens:
<pad> (index 0): για padding σε σταθερό μήκος
<unk> (index 1): για άγνωστες λέξεις
Η συνάρτηση numericalize μετατρέπει την κάθε λέξη στο αριθμητικό της index και άμα δεν υπάρχει unK
Έπειτα γίνεται η Φόρτωση IMDB μέσω torchtext και τα μετατρέπει α labels: 1→0 (neg), 2→1 (pos)
και προετοιμάζουμε datasets (prepare_datasets).Κόβει τα κείμενα στις πρώτες 200 λέξεις (για ταχύτητα).Χτίζει vocabulary από το train set
Μετατρέπει όλα τα κείμενα σε sequences αριθμών.
Τέλος το αρχείο αυτό ομαδοποιεί τα samples σε batches και κάνει padding ώστε τα sequences να εχουν ίδιο μήκος(collate_fn)

Το αρχείο glove_loader.py

Αρχικοποιεί τυχαία έναν πίνακα embeddings (vocab_size × 100)
Διαβάζει το αρχείο glove.6B.100d.txt
Για κάθε λέξη που υπάρχει στο vocab μας, αντικαθιστά το τυχαίο vector με το GloVe vector
Λέξεις που δεν υπάρχουν στο GloVe κρατάνε τυχαίες τιμές
GloVe embeddings:

Προεκπαιδευμένα word vectors από Wikipedia/Common Crawl
Κάθε λέξη αναπαρίσταται ως 100-διάστατο vector
Παρόμοιες λέξεις έχουν παρόμοια vectors (π.χ. "good" ≈ "great")

Στο αρχείο rnn_model.py

ορίζεται η αρχιτεκτονική.Δημιουργεί embedding layer με μέγεθος vocab_size × embed_dim.Αντιγράφει τα GloVe weights
padding_idx=0: αγνοεί τα padding tokens
freeze_embeddings: αν True, τα embeddings δεν ενημερώνονται κατά το training
GRU (Gated Recurrent Unit)είναι ο τύπος RNN που μαθαίνει μακροπρόθεσμες εξαρτήσεις
hidden_size=128: μέγεθος hidden state
bidirectional=True: διαβάζει το κείμενο και από τις δύο κατευθύνσεις (αρχή→τέλος και τέλος→αρχή)
num_layers: πόσα GRU layers στοιβάζονται (stacked)
dropout: regularization μεταξύ layers

Το classifier head είναι ένα MLP (Multi-Layer Perceptron)
που παίρνει την έξοδο του GRU και την μετατρέπει σε πρόβλεψη κλάσης.

Το Classifier Head έχει τα εξής χαρακτηριστικά
hidden_size * 2 γιατί είναι bidirectional (forward + backward)
Fully connected layer: 256 → 128
ReLU activation
Dropout για regularization
Output layer: 128 → 2 (neg/pos)

Τέλος η ροή δεδομένων γίνεται ώς εξής:

Embedding: μετατρέπει word indices σε vectors
GRU: επεξεργάζεται την ακολουθία, παράγει hidden states για κάθε θέση
Global Max Pooling: παίρνει το μέγιστο από όλες τις θέσεις (συνοψίζει όλο το κείμενο)
Classifier: παράγει τις τελικές πιθανότητες για κάθε κλάση

Το train_rnn.py

(Πρέπει να σημειωθεί πως έκανε αρκέτες ώρες στην αρχή να γίνει train απο 5+,γιαυτό μπορεί
να υπάρχει μήνυμα Using CPu καθώς είδα αμα χρησιμοποιεί GPU μπορεί να πάει πιο γρήγορα)

Φορτώνει και προετοιμάζει τα δεδομένα
Φορτώνει τα GloVe embedding
Έπειτα χωρίζει το train set σε 80% train / 20% dev
Το dev set χρησιμοποιείται για να επιλέξουμε την καλύτερη εποχή και χρησιμοποιούνται
DataLoaders με batch_size=128: επεξεργάζεται 128 κείμενα κάθε φορά
shuffle=True: ανακατεύει τα train data κάθε epoch
Το μοντέλο καθαυτό
CrossEntropyLoss: loss function για classification
Adam optimizer: αλγόριθμος βελτιστοποίησης (καλύτερος από απλό SGD)
lr=1e-3: learning rate
Στην συνέχεια για κάθε εποχή μεσα στην λούπα θα κάνει:
Για κάθε batch, κάνει forward pass
Υπολογίζει το loss
Κάνει backpropagation
Ενημερώνει τα weights.
Μετά την λούπα Αξιολογεί στο dev set μετά από κάθε epoch
Αποθηκεύει το μοντέλο με το χαμηλότερο dev loss
Τέλος παράγει γραφήματα με τα train/dev losses ανά epoch

Το τελευταίο αρχείο (evaluate_rnn.py)

αξιολογέι το μοντέλο στο test set.Φορτώνει 25k reviews,vocabulary embeddings,weights του καλύτερου μοντέλου.Κάνει predictions χωρίς να υπολογίζει gradients (torch.no_grad()) argmax: επιλέγει την κλάση με την υψηλότερη πιθανότητα.Τέλος Εκτυπώνει precision, recall, F1 για κάθε κλάση,Micro και macro averages

Υπερπαράμετροι Μέρους Β

embed_dim Τιμή = 100 και είναι η δίασταση GlOVe embeddings
hidden_size = 128 και είναι το Μέγεθος GRU hidden state
num_layers = 1 Αριθμός GRU layers
bidirectional = true σημαίνει οτι είναι Διλής κατεύθυνσης
dropout = 0.3
batch_size= 128 και είναι το μέγεθος batch
learning_rate= 0.001 που ειναι το learning rate Adam
max_seq_len = 200 Μέγισοτ μήκος sequence
min_freq = 2 ελάχιστη συχνότητα λέξης
epochs = 4 Αριθμός εποχών

freeze_embeddings = True Παγωμένα embeddings.

Μέρος Γ

Για την επίτευξη του τρίτου κομματιού δημιουργήθηκαν 3 αρχεία cnn_model.py,train_cnn.py,evaluate_cnn.py .

Εξωτερικές Βιβλιοθήκες

Πριν προχωρήσω στο τι είναι το καθέ ενα αρχείο, για αυτό το κομμάτι χρησιμοποιήθηκαν οι εξωτερικές βιβλιοθήκες:
PyTorch Cnn layers,training,loop,optimizer
torchtext Φόρτωση FashionMNIST dataset, προεκπαιδευμένο ResNet18, data augmentation transforms
NumPy Μετατροπή tensors σε arrays για metrics
matplotlib Γραφήματα loss curves
joblib Αποθήκευση/φόρτωση vocab.pk
scikit-learn classification_report για metrics

ο FashionMNIST είναι ένα dataset με 70.000 grayscale εικόνες ρούχων/αξεσουάρ:
60.000 train images,10.000 test images,Μέγεθος: 28×28 pixels,10 κατηγορίες

Το αρχείο cnn_model.py

είναι η αρχιτεκτονική του CNN.Φορτώνει Προεκπαιδευμένου ResNet18 (NN με 18 layers,Χρησιμοποιεί "residual connections" (skip connections)).Παρόλα αυτά δημιουργείται το πρόβλημα ότι ο ResNet18 περιμένει RGB εικόνες (3 κανάλια), αλλά το FashionMNIST είναι grayscale (1 κανάλι).Γιαυτό αντικαθιστούμε το πρώτο convolutional layer
Από Conv2d(3, 64, ...) σε Conv2d(1, 64, ...)
Αρχικοποιούμε τα weights παίρνοντας τον μέσο όρο των RGB weights.Πρέπει να σημειωθεί πως Αν freeze_backbone=True, τα weights του ResNet δεν ενημερώνονταικαι κάνει πίο γρήγορο το training.Τέλος υπάρχει ενα Custom MLP head.Εφόσον το αρχικό ResNet18 έχει fc layer για 1000 κλάσεις (ImageNet),το αντικαθιστούμε με δικό μας MLP για 10 κλάσεις.Η Αρχιτεκτονική μετατρέπεται: 512 → 256 → ReLU → Dropout → 10.Τέλος για το αρχείο αυτό περνάει μέσω της forward την εικόνα μέσα απο το ResNet που κάνει όλητην δουλεία

Το αρχειο train.cnn

Αυτό το αρχείο εκτελεί την εκπαίδευση του μοντέλου.Γίνεται Data augmentation ,δηλαδή δημιουργεί τεχνητές παραλλαγές των εικόνων:
RandomHorizontalFlip() Αναστρέφει οριζόντια με 50% πιθανότητα
RandomRotation(10) Περιστρέφει τυχαία ±10 μοίρες
RandomAffine(translate=(0.1, 0.1)) Μετακινεί τυχαία ±10%
ToTensor()Μετατρέπει σε PyTorch tensor
Normalize((0.5,), (0.5,)) Κανονικοποίηση στο [-1, 1]
Augmentation γίνεται με σκοπό να αυξηθέι η ποικιλία των training data,μειώνεται το overfitting και το μοντέλο μαθαίνει να αγνωρίζει αντικείμενα ανεξάρτητα απο θέση ή περιστροφή.Για dev/test όμως δεν κάνουμε augmentation
Στην συνέχεια, φορτώνουμε dataests και εφαρμόζει τα transforms.Γίνεται χωρισμός train set se 80% train/ 20% dev και στο dev set χρησιμοποιεί test_transform ,δηλαδή χωρίς augmentationΚάθε φορα επεξεργάζεται 64 εικόνες και τα ανακατέυει τα train data σε κάθε epoch.Το μοντέλο καθαυτό χαρακτηρίζεται από
freeze_backbone=False: Fine-tune όλο το ResNet
CrossEntropyLoss: Loss function για multi-class classification
Adam optimizer με learning rate 0.0001
Σε κάθε εποχή
model.train(): Ενεργοποιεί dropout και batch normalization σε train mode
Για κάθε batch:
Forward pass: out = model(x)
Υπολογισμός loss
Backward pass: loss.backward()
Ενημέρωση weights: optimizer.step()
Στην συνέχεια,απενεργοποιεί dropout μέσω της model.eval() , αποθηκέυει το μοντέλο με το χαμηλότερο
dev loss και παράγει γράφηματα με train/dev losses ανά εποχή

Το αρχείο evaluate_cnn.py

 ειναι το μοντέλο στο test set.Φορτώνει 10κ test images και χρησιμοποιεί test_transform (μόνο normalization),φορτώνει(δημιουργεί)το μοντέλο καθώς και τα weights απο το καλύτερο checkpoint και κάνει evaluation.Ταυτόχρονα για κάθε batch κάνει predictions και με το argmax επιλέγει την κλάση με τουψηλότερο σκορ,αλλά συλλέγει ολα τα labels και predictionsΤέλος Εκτυπώνει precision, recall, F1 για κάθε μία από τις 10 κλάσεις
Micro και macro averages

Συνολική accuracy

Υπερπαράμετροι Μέρους Γ

backbone=ResNet18 ,Προεκπαιδευμένο CNN
pretrained=ImageNet ,Weights από ImageNet
freeze_backbone=False ,Fine-tune όλο το δίκτυο
hidden_size (MLP)= 256 ,Νευρώνες στο hidden layer
dropout=0.3, Regularization
batch_size=64 ,Μέγεθος batch
learning_rate=0.0001 ,Learning rate για Adam
epochs=10 ,Αριθμός εποχών
augmentation=flip, rotate, translate είναι Data augmentation
normalization=(0.5), (0.5) Mean, std για normalize .

Φωτογραφίες/Πινακάκια

Μέρος Α

TRAIN LABELS: [10000 10000]

DEV LABELS: [2500 2500]

=== BernoulliNB ===

Precision: 0.862

Recall: 0.834

F1 score: 0.847

=== RandomForest ===

Precision: 0.857

Recall: 0.818

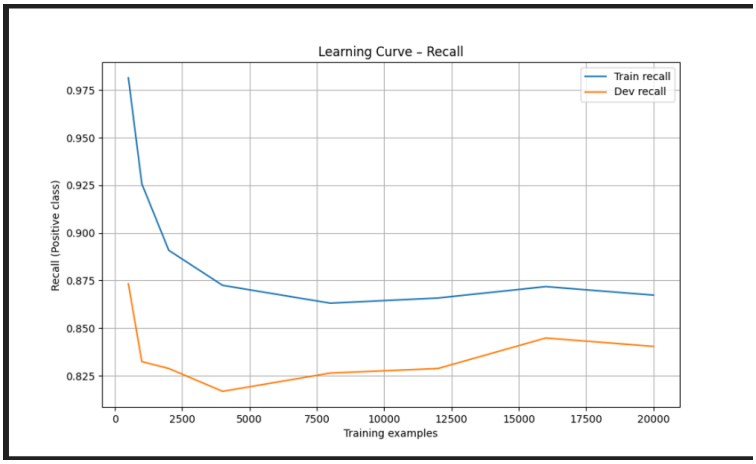
F1 score: 0.837

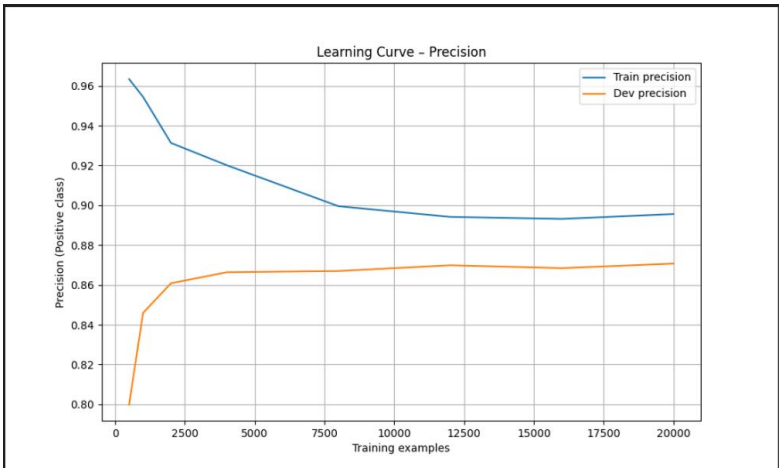
=== LogReg ===

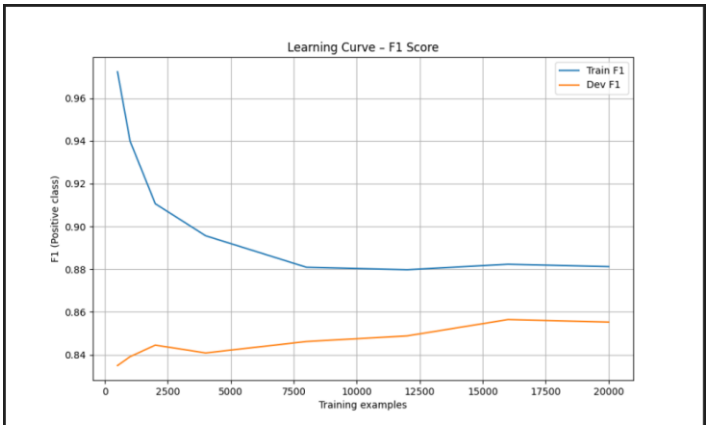
Precision: 0.867

Recall: 0.869

F1 score: 0.868







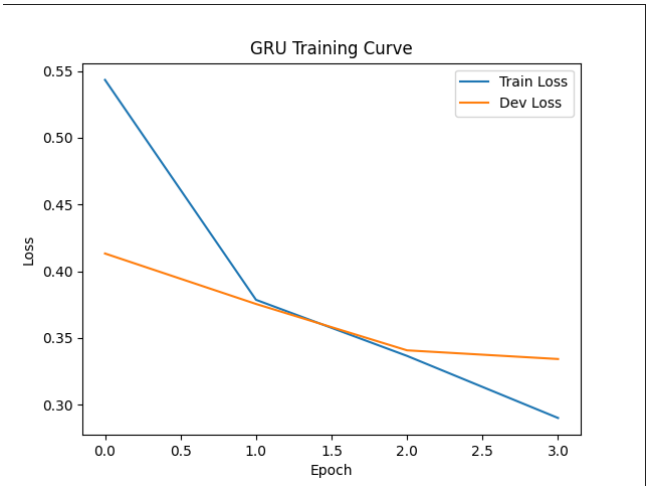
Μέρος Β

```
=== TEST SET RESULTS ===
```

```
              precision    recall  f1-score   support

         neg     0.8816    0.8610    0.8712     12500
         pos     0.8642    0.8844    0.8742     12500

    accuracy                         0.8727     25000
   macro avg     0.8729    0.8727    0.8727     25000
weighted avg     0.8729    0.8727    0.8727     25000
```



Μερος Γ

```
=== TEST SET RESULTS ===

              precision    recall  f1-score   support

 T-shirt/top     0.8537    0.8810    0.8671      1000
     Trouser     0.9940    0.9890    0.9915      1000
    Pullover     0.8691    0.8900    0.8794      1000
       Dress     0.9127    0.9100    0.9114      1000
        Coat     0.8705    0.8470    0.8586      1000
      Sandal     0.9820    0.9820    0.9820      1000
       Shirt     0.7796    0.7500    0.7645      1000
     Sneaker     0.9677    0.9600    0.9639      1000
         Bag     0.9754    0.9930    0.9841      1000
  Ankle boot     0.9643    0.9710    0.9676      1000

    accuracy                         0.9173     10000
```

```
   macro avg     0.9169    0.9173    0.9170     10000
weighted avg     0.9169    0.9173    0.9170     10000
```