ΕΠΕΞΕΡΓΑΣΙΑ ΦΥΣΙΚΗΣ ΓΛΩΣΣΑΣ - Εργασία 2

Γ. Text Classification with RNNs

Μάθημα: Επεξεργασία Φυσικής Γλώσσας

Συγγραφέας: Ιωάννης Κουτσούκης

Εκδόσεις: 2025-04-25 | v.0.0.1

🧱 Προετοιμασία Περιβάλλοντος

Το περιβάλλον υλοποιήθηκε σε **Mac με επεξεργαστή Apple Silicon (M1/M2/M3)**, με στόχο την αξιοποίηση των διαθέσιμων **υπολογιστικών πυρήνων GPU ή Neural Engine** μέσω του `torch.backends.mps`.

Για λόγους συμβατότητας και αποφυγής σφαλμάτων, επιλέχθηκαν οι **συγκεκριμένες εκδόσεις** των βιβλιοθηκών `torch` και `torchtext`. Η χρήση παλαιότερων ή νεότερων εκδόσεων ενδέχεται να οδηγήσει σε runtime conflicts, ειδικά σε Apple-based συστήματα.

In [155…

```
%pip install datasets numpy matplotlib --quiet
```

```
Note: you may need to restart the kernel to use updated packages.
```

In [ ]:

```
%pip install numpy==1.24.4 --force-reinstall --quiet
```

```
Collecting numpy==1.24.4
  Using cached numpy-1.24.4-cp311-cp311-macosx_11_0_arm64.whl.metadata (5.
6 kB)
Using cached numpy-1.24.4-cp311-cp311-macosx_11_0_arm64.whl (13.8 MB)
Installing collected packages: numpy
  Attempting uninstall: numpy
    Found existing installation: numpy 1.24.4
    Uninstalling numpy-1.24.4:
      Successfully uninstalled numpy-1.24.4
Successfully installed numpy-1.24.4
Note: you may need to restart the kernel to use updated packages.
```

In [156…

```
%pip install torch==2.1.0 torchtext==0.16.0  --quiet # οι υπολοιπες εκδοσ
```

```
Note: you may need to restart the kernel to use updated packages.
```

In [158…

```
import torch

if torch.backends.mps.is_available():
    device = torch.device("mps")
    print("🔥 Χρησιμοποιείται συσκευή: Apple Silicon GPU (MPS)")
elif torch.cuda.is_available():
    device = torch.device("cuda")
```

```
    print("⚡ Χρησιμοποιείται συσκευή: NVIDIA CUDA")
else:
    device = torch.device("cpu")
    print("💻 Χρησιμοποιείται συσκευή: CPU")
```

🔥 Χρησιμοποιείται συσκευή: Apple Silicon GPU (MPS)

📚 Εισαγωγή Βιβλιοθηκών

Οι κύριες βιβλιοθήκες που χρησιμοποιήθηκαν είναι:

- torch, torchtext: για την κατασκευή και εκπαίδευση του νευρωνικού μοντέλου.
- datasets: για εύκολη φόρτωση του IMDB dataset και άλλων πηγών.
- numpy, matplotlib, collections, re: για υποστήριξη, οπτικοποίηση και προεπεξεργασία.

In [ ]:
```
import torch.nn as nn
from torch.utils.data import DataLoader, Dataset

from datasets import load_dataset
import numpy as np
from collections import Counter
import re
import random
```

📥 Φόρτωση Δεδομένων – AG News Dataset

Το AG News dataset που χρησιμοποιήθηκε προέρχεται από την πηγή: Kaggle - AG News Classification Dataset

Αποτελείται από δύο αρχεία `.csv`:

- `train.csv` με 120.000 δείγματα
- `test.csv` με 7.600 δείγματα

Κάθε εγγραφή περιλαμβάνει:

- Κατηγορία (`Class Index`)
- Τίτλο της είδησης (`Title`)
- Σύντομη περιγραφή (`Description`)

Οι τίτλοι και περιγραφές συγχωνεύονται σε ένα ενιαίο string ώστε να χρησιμοποιηθούν ως είσοδοι στο μοντέλο.

In [194…
```
import pandas as pd

# Φόρτωση των CSV αρχείων
train_data = pd.read_csv('train.csv')
test_data = pd.read_csv('test.csv')

# Δημιουργία datasets (label, κείμενο)
train_dataset = [(label, train_data['Title'][i] + ' ' + train_data['Descr
test_dataset = [(label, test_data['Title'][i] + ' ' + test_data['Descript
```

```
print(f"Training samples: {len(train_dataset)}")
print(f"Test samples: {len(test_dataset)}")
```

```
Training samples: 120000
Test samples: 7600
```

✂️ Tokenization – Basic English Tokenizer

Για την μετατροπή του κειμένου σε tokens χρησιμοποιήθηκε ο ενσωματωμένος tokenizer της βιβλιοθήκης torchtext, συγκεκριμένα η μέθοδος basic_english.

Αυτός ο tokenizer:

- Μετατρέπει όλα τα γράμματα σε πεζά
- Αφαιρεί σημεία στίξης
- Διαχωρίζει λέξεις με βάση τα κενά

In [160…

```
#device = torch.device("cpu")

from torchtext.data import get_tokenizer

# Basic English tokenizer (lowercase, αφαιρεί σημεία στίξης κλπ)
tokenizer = get_tokenizer("basic_english")
```

📘 Δημιουργία Λεξιλογίου (Vocabulary)

Σύμφωνα με τις οδηγίες της εκφώνησης, δημιουργείται ένα λεξιλόγιο (`vocab`) που περιλαμβάνει:

- Όλες τις λέξεις που εμφανίζονται τουλάχιστον **10 φορές** στα training και test δεδομένα.
- Τα ειδικά tokens:
  - `<PAD>` για padding ακολουθιών
  - `<UNK>` για άγνωστες λέξεις (out-of-vocabulary)

Η διαδικασία περιλαμβάνει χρήση της συνάρτησης `build_vocab_from_iterator` από τη βιβλιοθήκη `torchtext`.

In [161…

```
from torchtext.vocab import build_vocab_from_iterator

# Συνάρτηση που παράγει λίστα tokenized λέξεων
def build_vocabulary(datasets):
    for dataset in datasets:
        for _, text in dataset:
            yield tokenizer(text)

# Δημιουργία λεξιλογίου από train + test dataset
vocab = build_vocab_from_iterator(
    build_vocabulary([train_dataset, test_dataset]),
    min_freq=10,
    specials=["<PAD>", "<UNK>"]
)
```

```python
# Ορισμός default index σε <UNK> (unknown words)
vocab.set_default_index(vocab["<UNK>"])

print(f"Vocabulary size: {len(vocab)}")
```

```
Vocabulary size: 21254
```

⚙️ Δημιουργία Συνάρτησης `collate_batch`

Για την εκπαίδευση του μοντέλου με `DataLoader`, απαιτείται μια custom `collate_fn` συνάρτηση που μετατρέπει κάθε batch από raw κείμενα και labels σε έτοιμα tensors για το μοντέλο.

Η παρακάτω συνάρτηση `collate_batch()` υλοποιεί:

- Tokenization των κειμένων
- Μετατροπή σε indices μέσω του λεξιλογίου (`vocab`)
- Truncation ή Padding των ακολουθιών σε σταθερό μήκος
- Optionally: αφαίρεση -1 από labels, αν ξεκινούν από 1 (π.χ. AGNews)

In [162…

```python
import torch

MAX_WORDS = 25

def collate_batch(batch, shift_labels=True):
    Y, X = list(zip(*batch))
    Y = torch.tensor(Y)

    if shift_labels:
        Y = Y - 1  # αλλαγή για datasets όπως AGNews
    else:
        Y


    # Tokenization + μετατροπή σε indices
    X = [vocab(tokenizer(text)) for text in X]

    # Truncation/Padding σε μήκος MAX_WORDS
    X = [
        tokens + [vocab['<PAD>']] * (MAX_WORDS - len(tokens)) if len(toke
        for tokens in X
    ]

    return torch.tensor(X, dtype=torch.int64).to(device), Y.to(device)
```

📦 Δημιουργία DataLoaders

Για τη διαδικασία εκπαίδευσης και αξιολόγησης χρησιμοποιούνται αντικείμενα `DataLoader`, τα οποία αναλαμβάνουν:

- Τη δημιουργία batches στα δεδομένα
- Τη χρήση της custom συνάρτησης `collate_batch` για tokenization και padding

- Shuffle των δεδομένων μόνο στο training set
- Επιτάχυνση μέσω lazy-loading, batch-wise μεταφορά σε GPU/MPS

Η τιμή του `BATCH_SIZE` ορίστηκε σε 1024 για ταχύτητα και επαρκή στατιστική ποικιλία σε κάθε batch.

In [163…

```
from torch.utils.data import DataLoader

# HYPER-PARAMETERS
BATCH_SIZE = 1024

# DataLoader για training set
train_loader = DataLoader(
    train_dataset,
    batch_size=BATCH_SIZE,
    shuffle=True,
    collate_fn=collate_batch
)

# DataLoader για test set
test_loader = DataLoader(
    test_dataset,
    batch_size=BATCH_SIZE,
    shuffle=False,
    collate_fn=collate_batch
)
```

🧠 Ορισμός Μοντέλου – `RNNClassifier` (AGNews & IMDB)

Η κλάση `RNNClassifier` αποτελεί τον πυρήνα του μοντέλου και έχει σχεδιαστεί με στόχο:

- Την υποστήριξη διαφορετικών τύπων RNN (`RNN` , `LSTM`)
- Την επιλογή αριθμού στρωμάτων και κατεύθυνσης (μονοκατευθυντικό ή bidirectional)
- Τη δυνατότητα εισαγωγής προεκπαιδευμένων embeddings
- Την επιλογή να "παγώσουν" (freeze) τα embeddings ή να συνεχίσουν να μαθαίνουν

In [164…

```
import torch.nn as nn
import torch.nn.functional as F

class RNNClassifier(nn.Module):
    def __init__(self,
                 vocab_size,
                 embedding_dim,
                 hidden_dim,
                 output_dim,
                 rnn_type="rnn",          # "rnn" ή "lstm"
                 num_layers=1,
                 bidirectional=False,
                 pretrained_embeddings=None,    # προσθήκη ερωτήματος 4
                 freeze_embeddings=False        # προσθήκη ερωτήματος 5
```

```
                  ):

        super(RNNClassifier, self).__init__()

        self.embedding = nn.Embedding(vocab_size, embedding_dim)

        if pretrained_embeddings is not None:                          #
            self.embedding.weight.data.copy_(pretrained_embeddings)     #

        if freeze_embeddings:                                     # προσθήκη ερ
            self.embedding.weight.requires_grad = False           # προσθήκη ερ

        if rnn_type == "rnn":
            self.rnn = nn.RNN(
                input_size=embedding_dim,
                hidden_size=hidden_dim,
                num_layers=num_layers,
                bidirectional=bidirectional,
                batch_first=True
            )
        elif rnn_type == "lstm":
            self.rnn = nn.LSTM(
                input_size=embedding_dim,
                hidden_size=hidden_dim,
                num_layers=num_layers,
                bidirectional=bidirectional,
                batch_first=True
            )
        else:
            raise ValueError("Unsupported rnn_type. Choose 'rnn' or 'lstm

        direction_factor = 2 if bidirectional else 1
        self.fc = nn.Linear(hidden_dim * direction_factor, output_dim)

    def forward(self, x):
        x = self.embedding(x)
        rnn_out, _ = self.rnn(x)
        out = self.fc(rnn_out[:, -1, :])  # παίρνουμε το τελευταίο output
        return F.softmax(out, dim=1)
```

🧪 Παραδείγματα Μοντέλων

Ακολουθούν παραδείγματα κατασκευής δύο βασικών μοντέλων με χρήση της παραμετρικής κλάσης `RNNClassifier`:

🔷 1-layer RNN

Το πιο απλό σενάριο: RNN με μία κρυφή στρώση, μονοκατευθυντικό.
Χρήσιμο για benchmarking ή εισαγωγικά πειράματα.

🔶 2-layer Bidirectional LSTM (BiLSTM)

Ισχυρότερο μοντέλο: LSTM, διπλής κατεύθυνσης (biLSTM), με 2 κρυφές στρώσεις.
Αξιοποιεί καλύτερα τη δομή του κειμένου καθώς διαβάζει και από τις δύο

κατευθύνσεις.

In [165…

```
# 1-layer RNN
model = RNNClassifier(
    vocab_size=len(vocab),
    embedding_dim=100,
    hidden_dim=64,
    output_dim=4,
    rnn_type="rnn",
    num_layers=1,
    bidirectional=False
).to(device)


# 2-layer BiLSTM
model = RNNClassifier(
    vocab_size=len(vocab),
    embedding_dim=100,
    hidden_dim=64,
    output_dim=4,
    rnn_type="lstm",
    num_layers=2,
    bidirectional=True
).to(device)
```

⚙️ Ορισμός Υπερπαραμέτρων

Οι βασικές υπερπαράμετροι που ορίζουν τη συμπεριφορά και την απόδοση του μοντέλου είναι οι εξής:

In [ ]:

```
MAX_WORDS = 25          # Μήκος token sequence (μετά από padding/truncatio
EPOCHS = 15             # Πλήρεις επαναλήψεις πάνω στο training set
LEARNING_RATE = 1e-3    # Ρυθμός μάθησης για τον optimizer
BATCH_SIZE = 1024       # Μέγεθος mini-batch (επηρεάζει μνήμη & ταχύτητα)
EMBEDDING_DIM = 100     # Διάσταση των word embeddings
HIDDEN_DIM = 64         # Διάσταση των hidden states του RNN
```

🧮 Συναρτήσεις Απώλειας και Βελτιστοποίησης

Για την εκπαίδευση των μοντέλων χρησιμοποιήθηκαν:

- Η συνάρτηση απώλειας `CrossEntropyLoss`, κατάλληλη για multi-class classification
- Ο βελτιστοποιητής `Adam`, γνωστός για γρήγορη και σταθερή σύγκλιση

In [167…

```
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)
```

📉 Συνάρτηση Αξιολόγησης – `evaluate_model`

Η συνάρτηση `evaluate_model()` αξιολογεί την απόδοση του μοντέλου σε test set (ή validation set), χωρίς να επηρεάζει τα gradients.

Επιστρέφει:

- Τη μέση τιμή απώλειας στο dataset (`total_loss / len(dataloader)`)
- Τα πραγματικά labels (`true_labels`)
- Τις προβλέψεις του μοντέλου (`pred_labels`)

In [168…

```
def evaluate_model(model, dataloader, loss_fn):
    model.eval()
    total_loss = 0
    true_labels = []
    pred_labels = []

    with torch.no_grad():
        for X_batch, y_batch in dataloader:
            outputs = model(X_batch)
            loss = loss_fn(outputs, y_batch)
            total_loss += loss.item()

            preds = outputs.argmax(dim=1)
            true_labels.append(y_batch.cpu())
            pred_labels.append(preds.cpu())

    true_labels = torch.cat(true_labels)
    pred_labels = torch.cat(pred_labels)

    return total_loss / len(dataloader), true_labels.numpy(), pred_labels
```

🏋️ Συνάρτηση Εκπαίδευσης – `train_model`

Η συνάρτηση `train_model()` υλοποιεί τη βασική ροή εκπαίδευσης ενός νευρωνικού μοντέλου για `epochs` επαναλήψεις.

Για κάθε εποχή πραγματοποιούνται:

- Εκπαίδευση πάνω στο training set
- Αξιολόγηση στο test set
- Εκτύπωση των βασικών μετρικών: train loss, val loss και val accuracy

In [169…

```
from tqdm import tqdm
from sklearn.metrics import accuracy_score

def train_model(model, train_loader, test_loader, optimizer, loss_fn, epo
    for epoch in range(1, epochs + 1):
        model.train()
        train_losses = []

        print(f"Epoch {epoch}")

        for X_batch, y_batch in tqdm(train_loader):
            optimizer.zero_grad()
            outputs = model(X_batch)
            loss = loss_fn(outputs, y_batch)
            loss.backward()
            optimizer.step()
```

```
            train_losses.append(loss.item())

        avg_train_loss = sum(train_losses) / len(train_losses)

        # Evaluation on test set
        val_loss, y_true, y_pred = evaluate_model(model, test_loader, los
        val_acc = accuracy_score(y_true, y_pred)

        print(f"Train Loss: {avg_train_loss:.4f} | Val Loss: {val_loss:.4
```

🚀 Εκπαίδευση RNN Μοντέλου (1-Layer RNN)

Παρακάτω παρουσιάζεται ένα πλήρες παράδειγμα εκπαίδευσης ενός απλού 1-layer RNN μοντέλου πάνω στο AGNews dataset:

- RNN (όχι LSTM)
- 1 στρώση (layer)
- Μονοκατευθυντικό (όχι bidirectional)
- 4 κατηγορίες εξόδου (World, Sports, Business, Sci/Tech)

In [ ]:
```
# Δημιουργία 1-layer RNN
model = RNNClassifier(
    vocab_size=len(vocab),
    embedding_dim=EMBEDDING_DIM,
    hidden_dim=HIDDEN_DIM,
    output_dim=4,             # 4 κατηγορίες (World, Sports, Business, Sci
    rnn_type="rnn",           # απλό RNN
    num_layers=1,             # 1 layer
    bidirectional=False       # όχι bidirectional
).to(device)

# Loss και Optimizer
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)

train_model(
    model=model,
    train_loader=train_loader,
    test_loader=test_loader,
    optimizer=optimizer,
    loss_fn=loss_fn,
    epochs=EPOCHS
)
```

🧪 Ορισμός Μοντέλων για Πειραματισμό

Όπως ζητείται στο Ερώτημα 1, εξετάζονται οι εξής έξι παραλλαγές RNN-based μοντέλων:

- Τύπος RNN: `RNN` ή `LSTM`
- Μονοκατευθυντικά ή διπλής κατεύθυνσης (bidirectional)
- Ένα ή δύο επίπεδα (layers)

Ο παρακάτω πίνακας περιέχει τα configurations όλων των μοντέλων:

In [173…

```
model_configs = [
    {"name": "1RNN",        "rnn_type": "rnn",  "num_layers": 1, "bidirect
    {"name": "1Bi-RNN",     "rnn_type": "rnn",  "num_layers": 1, "bidirect
    {"name": "2Bi-RNN",     "rnn_type": "rnn",  "num_layers": 2, "bidirect
    {"name": "1LSTM",       "rnn_type": "lstm", "num_layers": 1, "bidirect
    {"name": "1Bi-LSTM",    "rnn_type": "lstm", "num_layers": 1, "bidirect
    {"name": "2Bi-LSTM",    "rnn_type": "lstm", "num_layers": 2, "bidirect
]
```

🔄 Συνάρτηση Εκπαίδευσης για Κάθε Μοντέλο – `run_single_model()`

Η συνάρτηση `run_single_model()` αναλαμβάνει:

- Να χτίσει το μοντέλο με βάση ένα configuration (`config`)
- Να το εκπαιδεύσει για `EPOCHS` επαναλήψεις
- Να το αξιολογήσει
- Να επιστρέψει τα βασικά στατιστικά για πίνακες σύγκρισης και error analysis

🔧 Είσοδοι (Arguments):

Παράμετρος	Περιγραφή
`config`	Λεξικό με στοιχεία αρχιτεκτονικής (όνομα, layers κ.λπ.)
`pretrained_embeddings`	Embedding matrix (GloVe κ.λπ.) ή `None`
`freeze_embeddings`	Αν θα εκπαιδεύονται τα embeddings ή όχι
`vocab_to_use`	Το λεξιλόγιο που θα χρησιμοποιηθεί
`output_dim`	Αριθμός εξόδων (4 για AGNews, 2 για IMDB)
`train_loader`	DataLoader για training set

| `test_loader` | DataLoader για test set

In [174…

```
import time

def run_single_model(config,
                     pretrained_embeddings=None,  # προσθήκη ερωτήματος 4
                     freeze_embeddings=False,  # προσθήκη ερωτήματος 5
                     vocab_to_use=vocab,  # προσθήκη ερωτήματος 6
                     output_dim=4,            # προσθήκη ερωτήματος 6
                     train_loader=train_loader,         # προσθήκη ερωτήμ
                     test_loader=test_loader           # προσθήκη ερωτήμα
                     ):
    print(f"\n🔄 Training: {config['name']}")



    model = RNNClassifier(
        vocab_size=len(vocab_to_use),
```

```
        embedding_dim=EMBEDDING_DIM,
        hidden_dim=HIDDEN_DIM,
        output_dim=output_dim,
        rnn_type=config["rnn_type"],
        num_layers=config["num_layers"],
        bidirectional=config["bidirectional"],
        pretrained_embeddings=pretrained_embeddings,  # προσθήκη ερωτήματ
        freeze_embeddings=freeze_embeddings # προσθήκη ερωτήματος 5
    ).to(device)

    optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)
    loss_fn = nn.CrossEntropyLoss()

    start = time.time()
    train_model(model, train_loader, test_loader, optimizer, loss_fn, epo
    end = time.time()

    total_params = sum(p.numel() for p in model.parameters() if p.require
    val_loss, y_true, y_pred = evaluate_model(model, test_loader, loss_fn
    val_acc = accuracy_score(y_true, y_pred)

    return {
        "Model": config['name'],
        "Accuracy": round(val_acc * 100, 2),
        "Parameters": total_params,
        "Time (sec)": round((end - start) / EPOCHS, 2),
        "y_pred": y_pred   # <-- Προσθέτουμε τα predictions
    }
```

🧪 Εκτέλεση Όλων των Μοντέλων – `run_experiments()`

Η συνάρτηση `run_experiments()` χρησιμοποιείται για να εκτελέσει **όλα τα μοντέλα που καθορίζονται στον πίνακα `model_configs`**, και να επιστρέψει:

- Ένα DataFrame με μετρικές απόδοσης για κάθε μοντέλο
- Ένα dictionary με τα predicted labels (`y_pred`) κάθε μοντέλου (για ανάλυση σφαλμάτων)

In [175…

```
import pandas as pd

def run_experiments(configs, pretrained_embeddings=None, freeze_embedding
    results = []
    preds_dict = {}

    for cfg in configs:
        result = run_single_model(cfg, pretrained_embeddings=pretrained_e
        vocab_to_use=vocab_to_use,
        output_dim=output_dim,
        train_loader=train_loader,
        test_loader=test_loader
            ) # pretrained_embeddings προσθήκη ερωτήματος 4 | freeze_embe
        preds_dict[cfg['name']] = result.pop('y_pred')  # Αποθηκεύουμε y_
        results.append(result)
```

```
    df_results = pd.DataFrame(results)
    return df_results, preds_dict
```

Απαντήσεις Ερωτημάτων

📊 Ερώτημα Γ.1 – Εκτέλεση Πειραμάτων & Συμπλήρωση Πίνακα

Σε αυτό το ερώτημα εξετάζονται έξι διαφορετικά μοντέλα που διαφέρουν ως προς:

- Τον τύπο RNN (`RNN` ή `LSTM`)
- Το αν είναι μονοκατευθυντικά ή bidirectional
- Τον αριθμό των επιπέδων (layers): 1 ή 2

In [176…

```
results_table, preds_dict = run_experiments(model_configs)
```

```
🔄 Training: 1RNN
Epoch 1
100%|██████████| 118/118 [00:03<00:00, 36.01it/s]
Train Loss: 1.2895 | Val Loss: 1.1397 | Val Accuracy: 0.6053

Epoch 2
100%|██████████| 118/118 [00:03<00:00, 37.01it/s]
Train Loss: 1.0579 | Val Loss: 1.0076 | Val Accuracy: 0.7370

Epoch 3
100%|██████████| 118/118 [00:03<00:00, 37.33it/s]
Train Loss: 0.9659 | Val Loss: 0.9579 | Val Accuracy: 0.7862

Epoch 4
100%|██████████| 118/118 [00:03<00:00, 37.23it/s]
Train Loss: 0.9256 | Val Loss: 0.9394 | Val Accuracy: 0.8026

Epoch 5
100%|██████████| 118/118 [00:03<00:00, 30.15it/s]
Train Loss: 0.9030 | Val Loss: 0.9171 | Val Accuracy: 0.8253

Epoch 6
100%|██████████| 118/118 [00:03<00:00, 34.03it/s]
Train Loss: 0.8868 | Val Loss: 0.9066 | Val Accuracy: 0.8346

Epoch 7
100%|██████████| 118/118 [00:03<00:00, 36.72it/s]
Train Loss: 0.8743 | Val Loss: 0.8973 | Val Accuracy: 0.8449

Epoch 8
100%|██████████| 118/118 [00:03<00:00, 36.43it/s]
Train Loss: 0.8661 | Val Loss: 0.8969 | Val Accuracy: 0.8443

Epoch 9
100%|██████████| 118/118 [00:03<00:00, 36.89it/s]
```

```
Train Loss: 0.8579 | Val Loss: 0.8910 | Val Accuracy: 0.8503

Epoch 10
100%|██████████| 118/118 [00:03<00:00, 37.28it/s]
Train Loss: 0.8529 | Val Loss: 0.8847 | Val Accuracy: 0.8574

Epoch 11
100%|██████████| 118/118 [00:03<00:00, 37.08it/s]
Train Loss: 0.8477 | Val Loss: 0.8820 | Val Accuracy: 0.8596

Epoch 12
100%|██████████| 118/118 [00:03<00:00, 36.94it/s]
Train Loss: 0.8437 | Val Loss: 0.8817 | Val Accuracy: 0.8603

Epoch 13
100%|██████████| 118/118 [00:03<00:00, 35.07it/s]
Train Loss: 0.8398 | Val Loss: 0.8764 | Val Accuracy: 0.8646

Epoch 14
100%|██████████| 118/118 [00:03<00:00, 36.79it/s]
Train Loss: 0.8356 | Val Loss: 0.8784 | Val Accuracy: 0.8650

Epoch 15
100%|██████████| 118/118 [00:03<00:00, 35.67it/s]
Train Loss: 0.8329 | Val Loss: 0.8748 | Val Accuracy: 0.8664


🔄 Training: 1Bi-RNN
Epoch 1
100%|██████████| 118/118 [00:04<00:00, 23.67it/s]
Train Loss: 1.2968 | Val Loss: 1.1575 | Val Accuracy: 0.5787

Epoch 2
100%|██████████| 118/118 [00:04<00:00, 28.84it/s]
Train Loss: 1.0878 | Val Loss: 1.0124 | Val Accuracy: 0.7339

Epoch 3
100%|██████████| 118/118 [00:04<00:00, 28.66it/s]
Train Loss: 0.9724 | Val Loss: 0.9546 | Val Accuracy: 0.7920

Epoch 4
100%|██████████| 118/118 [00:04<00:00, 29.05it/s]
Train Loss: 0.9249 | Val Loss: 0.9286 | Val Accuracy: 0.8142

Epoch 5
100%|██████████| 118/118 [00:05<00:00, 22.48it/s]
Train Loss: 0.9008 | Val Loss: 0.9130 | Val Accuracy: 0.8295

Epoch 6
100%|██████████| 118/118 [00:04<00:00, 27.45it/s]
Train Loss: 0.8841 | Val Loss: 0.9030 | Val Accuracy: 0.8382

Epoch 7
100%|██████████| 118/118 [00:04<00:00, 27.15it/s]
```

```
Train Loss: 0.8716 | Val Loss: 0.8920 | Val Accuracy: 0.8492

Epoch 8
100%|██████████| 118/118 [00:04<00:00, 28.92it/s]
Train Loss: 0.8634 | Val Loss: 0.8890 | Val Accuracy: 0.8525

Epoch 9
100%|██████████| 118/118 [00:04<00:00, 29.10it/s]
Train Loss: 0.8548 | Val Loss: 0.8870 | Val Accuracy: 0.8536

Epoch 10
100%|██████████| 118/118 [00:04<00:00, 29.07it/s]
Train Loss: 0.8489 | Val Loss: 0.8838 | Val Accuracy: 0.8561

Epoch 11
100%|██████████| 118/118 [00:04<00:00, 28.56it/s]
Train Loss: 0.8442 | Val Loss: 0.8770 | Val Accuracy: 0.8638

Epoch 12
100%|██████████| 118/118 [00:04<00:00, 27.70it/s]
Train Loss: 0.8395 | Val Loss: 0.8821 | Val Accuracy: 0.8588

Epoch 13
100%|██████████| 118/118 [00:04<00:00, 26.49it/s]
Train Loss: 0.8368 | Val Loss: 0.8732 | Val Accuracy: 0.8667

Epoch 14
100%|██████████| 118/118 [00:04<00:00, 26.94it/s]
Train Loss: 0.8348 | Val Loss: 0.8709 | Val Accuracy: 0.8700

Epoch 15
100%|██████████| 118/118 [00:04<00:00, 29.03it/s]
Train Loss: 0.8303 | Val Loss: 0.8731 | Val Accuracy: 0.8680


🔄 Training: 2Bi-RNN
Epoch 1
100%|██████████| 118/118 [00:06<00:00, 19.38it/s]
Train Loss: 1.2619 | Val Loss: 1.1031 | Val Accuracy: 0.6339

Epoch 2
100%|██████████| 118/118 [00:06<00:00, 19.22it/s]
Train Loss: 1.0447 | Val Loss: 0.9992 | Val Accuracy: 0.7399

Epoch 3
100%|██████████| 118/118 [00:06<00:00, 18.37it/s]
Train Loss: 0.9715 | Val Loss: 0.9635 | Val Accuracy: 0.7784

Epoch 4
100%|██████████| 118/118 [00:06<00:00, 19.28it/s]
Train Loss: 0.9361 | Val Loss: 0.9345 | Val Accuracy: 0.8072

Epoch 5
100%|██████████| 118/118 [00:06<00:00, 19.58it/s]
```

```
Train Loss: 0.9117 | Val Loss: 0.9255 | Val Accuracy: 0.8129

Epoch 6
100%|██████████| 118/118 [00:06<00:00, 18.40it/s]
Train Loss: 0.8998 | Val Loss: 0.9091 | Val Accuracy: 0.8308

Epoch 7
100%|██████████| 118/118 [00:06<00:00, 19.36it/s]
Train Loss: 0.8921 | Val Loss: 0.9082 | Val Accuracy: 0.8308

Epoch 8
100%|██████████| 118/118 [00:06<00:00, 19.44it/s]
Train Loss: 0.8798 | Val Loss: 0.9030 | Val Accuracy: 0.8353

Epoch 9
100%|██████████| 118/118 [00:07<00:00, 16.22it/s]
Train Loss: 0.8760 | Val Loss: 0.9025 | Val Accuracy: 0.8371

Epoch 10
100%|██████████| 118/118 [00:08<00:00, 13.25it/s]
Train Loss: 0.8725 | Val Loss: 0.9068 | Val Accuracy: 0.8320

Epoch 11
100%|██████████| 118/118 [00:08<00:00, 13.50it/s]
Train Loss: 0.8675 | Val Loss: 0.8929 | Val Accuracy: 0.8478

Epoch 12
100%|██████████| 118/118 [00:06<00:00, 19.41it/s]
Train Loss: 0.8701 | Val Loss: 0.8934 | Val Accuracy: 0.8474

Epoch 13
100%|██████████| 118/118 [00:06<00:00, 18.52it/s]
Train Loss: 0.8599 | Val Loss: 0.8854 | Val Accuracy: 0.8557

Epoch 14
100%|██████████| 118/118 [00:06<00:00, 19.47it/s]
Train Loss: 0.8581 | Val Loss: 0.8798 | Val Accuracy: 0.8611

Epoch 15
100%|██████████| 118/118 [00:06<00:00, 16.89it/s]
Train Loss: 0.8622 | Val Loss: 0.8841 | Val Accuracy: 0.8567


🔄 Training: 1LSTM
Epoch 1
100%|██████████| 118/118 [00:04<00:00, 28.86it/s]
Train Loss: 1.2385 | Val Loss: 1.0363 | Val Accuracy: 0.7105

Epoch 2
100%|██████████| 118/118 [00:04<00:00, 28.78it/s]
Train Loss: 0.9723 | Val Loss: 0.9350 | Val Accuracy: 0.8121

Epoch 3
100%|██████████| 118/118 [00:03<00:00, 30.24it/s]
```

```
Train Loss: 0.9088 | Val Loss: 0.9013 | Val Accuracy: 0.8405

Epoch 4
100%|██████████| 118/118 [00:04<00:00, 28.78it/s]
Train Loss: 0.8810 | Val Loss: 0.8861 | Val Accuracy: 0.8568

Epoch 5
100%|██████████| 118/118 [00:04<00:00, 25.36it/s]
Train Loss: 0.8619 | Val Loss: 0.8808 | Val Accuracy: 0.8613

Epoch 6
100%|██████████| 118/118 [00:04<00:00, 27.54it/s]
Train Loss: 0.8509 | Val Loss: 0.8725 | Val Accuracy: 0.8695

Epoch 7
100%|██████████| 118/118 [00:04<00:00, 28.62it/s]
Train Loss: 0.8412 | Val Loss: 0.8702 | Val Accuracy: 0.8716

Epoch 8
100%|██████████| 118/118 [00:04<00:00, 28.48it/s]
Train Loss: 0.8345 | Val Loss: 0.8705 | Val Accuracy: 0.8718

Epoch 9
100%|██████████| 118/118 [00:03<00:00, 31.00it/s]
Train Loss: 0.8287 | Val Loss: 0.8631 | Val Accuracy: 0.8787

Epoch 10
100%|██████████| 118/118 [00:03<00:00, 29.91it/s]
Train Loss: 0.8231 | Val Loss: 0.8638 | Val Accuracy: 0.8792

Epoch 11
100%|██████████| 118/118 [00:03<00:00, 29.57it/s]
Train Loss: 0.8206 | Val Loss: 0.8612 | Val Accuracy: 0.8805

Epoch 12
100%|██████████| 118/118 [00:03<00:00, 30.24it/s]
Train Loss: 0.8167 | Val Loss: 0.8581 | Val Accuracy: 0.8833

Epoch 13
100%|██████████| 118/118 [00:03<00:00, 30.61it/s]
Train Loss: 0.8135 | Val Loss: 0.8582 | Val Accuracy: 0.8846

Epoch 14
100%|██████████| 118/118 [00:04<00:00, 27.96it/s]
Train Loss: 0.8113 | Val Loss: 0.8567 | Val Accuracy: 0.8843

Epoch 15
100%|██████████| 118/118 [00:04<00:00, 29.11it/s]
Train Loss: 0.8095 | Val Loss: 0.8565 | Val Accuracy: 0.8859


🔄 Training: 1Bi-LSTM
Epoch 1
100%|██████████| 118/118 [00:04<00:00, 27.17it/s]
```

```
Train Loss: 1.2642 | Val Loss: 1.0614 | Val Accuracy: 0.6859

Epoch 2
100%|██████████| 118/118 [00:04<00:00, 27.52it/s]
Train Loss: 0.9814 | Val Loss: 0.9370 | Val Accuracy: 0.8067

Epoch 3
100%|██████████| 118/118 [00:04<00:00, 28.45it/s]
Train Loss: 0.9100 | Val Loss: 0.9086 | Val Accuracy: 0.8332

Epoch 4
100%|██████████| 118/118 [00:04<00:00, 27.64it/s]
Train Loss: 0.8801 | Val Loss: 0.8891 | Val Accuracy: 0.8522

Epoch 5
100%|██████████| 118/118 [00:05<00:00, 20.07it/s]
Train Loss: 0.8622 | Val Loss: 0.8800 | Val Accuracy: 0.8622

Epoch 6
100%|██████████| 118/118 [00:05<00:00, 23.15it/s]
Train Loss: 0.8502 | Val Loss: 0.8757 | Val Accuracy: 0.8651

Epoch 7
100%|██████████| 118/118 [00:04<00:00, 27.40it/s]
Train Loss: 0.8420 | Val Loss: 0.8694 | Val Accuracy: 0.8728

Epoch 8
100%|██████████| 118/118 [00:04<00:00, 28.36it/s]
Train Loss: 0.8348 | Val Loss: 0.8680 | Val Accuracy: 0.8739

Epoch 9
100%|██████████| 118/118 [00:04<00:00, 27.71it/s]
Train Loss: 0.8288 | Val Loss: 0.8646 | Val Accuracy: 0.8755

Epoch 10
100%|██████████| 118/118 [00:05<00:00, 19.97it/s]
Train Loss: 0.8248 | Val Loss: 0.8633 | Val Accuracy: 0.8783

Epoch 11
100%|██████████| 118/118 [00:07<00:00, 15.27it/s]
Train Loss: 0.8202 | Val Loss: 0.8617 | Val Accuracy: 0.8797

Epoch 12
100%|██████████| 118/118 [00:04<00:00, 27.56it/s]
Train Loss: 0.8171 | Val Loss: 0.8635 | Val Accuracy: 0.8772

Epoch 13
100%|██████████| 118/118 [00:04<00:00, 28.32it/s]
Train Loss: 0.8148 | Val Loss: 0.8620 | Val Accuracy: 0.8793

Epoch 14
100%|██████████| 118/118 [00:04<00:00, 24.84it/s]
Train Loss: 0.8118 | Val Loss: 0.8591 | Val Accuracy: 0.8812

Epoch 15
100%|██████████| 118/118 [00:04<00:00, 28.04it/s]
```

```
Train Loss: 0.8092 | Val Loss: 0.8595 | Val Accuracy: 0.8812


🔄 Training: 2Bi-LSTM
Epoch 1
100%|██████████| 118/118 [00:10<00:00, 10.85it/s]
Train Loss: 1.1963 | Val Loss: 1.0044 | Val Accuracy: 0.7392

Epoch 2
100%|██████████| 118/118 [00:07<00:00, 15.49it/s]
Train Loss: 0.9465 | Val Loss: 0.9130 | Val Accuracy: 0.8289

Epoch 3
100%|██████████| 118/118 [00:06<00:00, 18.02it/s]
Train Loss: 0.8916 | Val Loss: 0.8974 | Val Accuracy: 0.8434

Epoch 4
100%|██████████| 118/118 [00:06<00:00, 18.15it/s]
Train Loss: 0.8684 | Val Loss: 0.8800 | Val Accuracy: 0.8600

Epoch 5
100%|██████████| 118/118 [00:11<00:00, 10.17it/s]
Train Loss: 0.8528 | Val Loss: 0.8752 | Val Accuracy: 0.8658

Epoch 6
100%|██████████| 118/118 [00:09<00:00, 12.58it/s]
Train Loss: 0.8421 | Val Loss: 0.8701 | Val Accuracy: 0.8709

Epoch 7
100%|██████████| 118/118 [00:06<00:00, 18.19it/s]
Train Loss: 0.8342 | Val Loss: 0.8684 | Val Accuracy: 0.8730

Epoch 8
100%|██████████| 118/118 [00:06<00:00, 18.00it/s]
Train Loss: 0.8283 | Val Loss: 0.8639 | Val Accuracy: 0.8779

Epoch 9
100%|██████████| 118/118 [00:08<00:00, 14.27it/s]
Train Loss: 0.8228 | Val Loss: 0.8601 | Val Accuracy: 0.8799

Epoch 10
100%|██████████| 118/118 [00:06<00:00, 17.86it/s]
Train Loss: 0.8191 | Val Loss: 0.8585 | Val Accuracy: 0.8841

Epoch 11
100%|██████████| 118/118 [00:06<00:00, 18.22it/s]
Train Loss: 0.8174 | Val Loss: 0.8601 | Val Accuracy: 0.8808

Epoch 12
100%|██████████| 118/118 [00:08<00:00, 13.80it/s]
Train Loss: 0.8134 | Val Loss: 0.8568 | Val Accuracy: 0.8837

Epoch 13
100%|██████████| 118/118 [00:12<00:00,  9.47it/s]
```

```
Train Loss: 0.8107 | Val Loss: 0.8607 | Val Accuracy: 0.8804

Epoch 14
100%|██████████| 118/118 [00:06<00:00, 18.06it/s]
Train Loss: 0.8101 | Val Loss: 0.8571 | Val Accuracy: 0.8838

Epoch 15
100%|██████████| 118/118 [00:06<00:00, 18.29it/s]
Train Loss: 0.8068 | Val Loss: 0.8579 | Val Accuracy: 0.8841
```

Κατασκευή Πίνακα Ερωτήματος Γ.1

In [179…

```python
# Transpose results to match the table format in the assignment
df_transposed = results_table.set_index("Model").T

# Round accuracy to 2 decimals, format numbers
df_display = pd.DataFrame({
    col: [
        f"{df_transposed.loc['Accuracy', col]:.2f}",
        f"{int(df_transposed.loc['Parameters', col]):,}",
        f"{df_transposed.loc['Time (sec)', col]:.2f}"
    ]
    for col in df_transposed.columns
}, index=["Accuracy", "Parameters", "Time cost"])

df_display
```

Out[179…

	1RNN	1Bi-RNN	2Bi-RNN	1LSTM	1Bi-LSTM	2Bi-LSTM
Accuracy	86.64	86.80	85.67	88.59	88.12	88.41
Parameters	2,136,284	2,147,164	2,171,996	2,168,156	2,210,908	2,310,236
Time cost	3.42	4.47	6.84	4.27	4.98	8.32

Ερώτημα Γ.1 – Συμπεράσματα

Από τη συγκριτική μελέτη των έξι μοντέλων (RNN & LSTM, με/χωρίς bidirectionality και 1 ή 2 στρώματα) προκύπτουν τα εξής γενικά συμπεράσματα:

- **Η αύξηση της ακρίβειας συνοδεύεται σταθερά από αύξηση του χρόνου εκπαίδευσης και της πολυπλοκότητας του μοντέλου** (παραμέτρων). Το μοντέλο `2Bi-LSTM`, για παράδειγμα, είχε το μεγαλύτερο accuracy αλλά και το υψηλότερο κόστος χρόνου και αριθμό παραμέτρων.

- **Η τελική ακρίβεια όλων των μοντέλων ήταν πολύ κοντά** (από 85.88% έως 88.59%). Αυτό σημαίνει πως δεν είναι εύκολο να εξαχθούν απόλυτα συμπεράσματα υπέρ κάποιας αρχιτεκτονικής.

- **Η χρήση bidirectional ροής και δεύτερου στρώματος δεν φαίνεται να προσφέρει σταθερά πλεονεκτήματα ως προς την ακρίβεια**. Ορισμένα Bi-models είχαν ελαφρώς χαμηλότερη ακρίβεια από τα απλά.

📌 Συμπερασματικά, η επιλογή του κατάλληλου μοντέλου εξαρτάται κυρίως από **τις ανάγκες του συστήματος**: αν προέχει η ακρίβεια ή η ταχύτητα/απλότητα.

🧩 Ερώτημα Γ.2 – Εντοπισμός Κοινών Λανθασμένων Προβλέψεων

Στόχος του ερωτήματος είναι να εντοπίσουμε περιπτώσεις όπου **όλα τα μοντέλα** αποτυγχάνουν να προβλέψουν σωστά την κατηγορία ενός κειμένου.

Δημιουργία DataFrame με Όλες τις Προβλέψεις

Για κάθε sample του test set, δημιουργούμε ένα νέο `DataFrame` που περιλαμβάνει:

- Το πλήρες κείμενο (Title + Description)
- Το ground truth label (0–3)
- Το κατηγορικό όνομα της κατηγορίας (`World` , `Sports` , κ.λπ.)
- Τις προβλέψεις κάθε μοντέλου (`1RNN` , `1Bi-RNN` , ...)

In [180…

```
# Ground truth labels (0–3)
y_true = test_data["Class Index"].values - 1

# Map index → category
label_names = ["World", "Sports", "Business", "Sci/Tech"]

# Δημιουργία βασικού DataFrame
df_errors = pd.DataFrame({
    "text": test_data['Title'] + ' ' + test_data['Description'],
    "true_label": y_true
})

# Προσθήκη προβλέψεων από όλα τα μοντέλα
for model_name, preds in preds_dict.items():
    df_errors[model_name] = preds

# Αντιστοιχία σε κατηγορία (text)
df_errors["true_category"] = df_errors["true_label"].apply(lambda i: labe
for model_name in preds_dict:
    df_errors[model_name] = df_errors[model_name].apply(lambda i: label_n
```

❌ Εντοπισμός Κοινών Λαθών

Ορίζεται ως κοινό λάθος όταν όλα τα μοντέλα διαφωνούν με την πραγματική κατηγορία:

In [182…

```
from collections import Counter


def majority_vote(row):
    preds = [row[m] for m in preds_dict]
    return Counter(preds).most_common(1)[0][0]
```

```
# Βρίσκουμε πού όλα τα μοντέλα κάνουν λάθος
all_wrong_mask = df_errors.apply(
    lambda row: all(row[m] != row["true_category"] for m in preds_dict),
    axis=1
)

# Φιλτράρουμε τα κοινά λάθη
df_all_wrong = df_errors[all_wrong_mask].copy()

# Υπολογίζουμε majority vote για κάθε κοινό λάθος
df_all_wrong["majority_vote"] = df_all_wrong.apply(majority_vote, axis=1)

print(f" * Συνολικά κοινά λάθη από όλα τα μοντέλα: {len(df_all_wrong)}")
print("\n * Κατανομή κοινών λαθών ανά κατηγορία:")
print(df_all_wrong["true_category"].value_counts())
```

```
 * Συνολικά κοινά λάθη από όλα τα μοντέλα: 333

 * Κατανομή κοινών λαθών ανά κατηγορία:
true_category
Business    125
World       116
Sci/Tech     77
Sports       15
Name: count, dtype: int64
```

In [183…

```
# Εμφάνιση πρώτων 10 κοινών λαθών με τις προβλέψεις
df_all_wrong[["text", "true_category", *preds_dict.keys(), "majority_vote
```

Out[183…

	text	true_category	1RNN	1Bi-RNN	2Bi-RNN	1LSTM	1Bi-LSTM	
79	Live: Olympics day four Richard Faulds and Ste...	World	Sports	Sports	Sports	Sports	Sports	
83	Intel to delay product aimed for high-definiti...	Business	Sci/Tech	Sci/Tech	Sci/Tech	Sci/Tech	Sci/Tech	S
88	U.S. Misses Cut in Olympic 100 Free ATHENS, Gr...	World	Sports	Sports	Sports	Sports	Sports	
89	Consumers Would Pay In Phone Proposal A propos...	Sci/Tech	Business	Business	Business	Business	Business	B
106	Stocks Climb on Drop in Consumer Prices NEW YO...	World	Business	Business	Business	Business	Business	B
110	Yahoo! Ups Ante for Small Businesses Web giant...	Business	Sci/Tech	Sci/Tech	Sci/Tech	Sci/Tech	Sci/Tech	S
120	Oil prices bubble to record high The price of ...	World	Business	Business	Business	Business	Business	B
154	Google Lowers Its IPO Price Range SAN JOSE, Ca...	World	Sci/Tech	Business	Sci/Tech	Business	Business	S
196	Stock Prices Climb Ahead of Google IPO NEW YOR...	World	Business	Business	Business	Business	Business	B

	text	true_category	1RNN	1Bi-RNN	2Bi-RNN	1LSTM	1Bi-LSTM
200	Strong Family Equals Strong Education Single m...	Sci/Tech	Business	World	Business	World	Business

Ερώτημα Γ.2 – Συμπεράσματα

Αναλύοντας τα δείγματα του test set στα οποία **όλα τα RNN-based μοντέλα έκαναν λάθος**, προκύπτουν τα εξής:

- ❌ **Συνολικά 333 περιπτώσεις** προβλέφθηκαν λανθασμένα από όλα τα μοντέλα.
- 📊 Η **κατανομή των κοινών λαθών** ανά κατηγορία έχει ιδιαίτερο ενδιαφέρον:
  - `World` : 125 λάθη
  - `Business` : 116 λάθη
  - `Sci/Tech` : 77 λάθη
  - `Sports` : **μόνο 15 λάθη**
- ⚽ Η κατηγορία `Sports` έχει **πολύ λιγότερα κοινά λάθη**, κάτι που υποδηλώνει ότι τα μοντέλα αναγνωρίζουν πιο εύκολα τα αθλητικά κείμενα.
- Στις περισσότερες περιπτώσεις που παρατηρήθηκε λάθος:
  - **Στη πλειοψηφία τους τα μοντέλα συμφώνησαν** στο λάθος (π.χ. προέβλεψαν "Business" αντί "World").
  - Η **πλειοψηφία των μοντέλων (majority vote)** απέδωσε το ίδιο λάθος label, υποδηλώνοντας κοινή αστοχία κατανόησης.
- Συγκριτικά με τα μοντέλα του Β' μέρους (Naive Bayes & SVM), παρατηρούμε:
  - Παρόμοια κατανομή λαθών.
  - Αντίστοιχα χαρακτηριστικά δυσκολίας στις κατηγορίες `World` και `Business`.

📌 Γενικό συμπέρασμα:

> Παρότι τα RNN-based μοντέλα είναι πιο εξελιγμένα, αποτυγχάνουν σε **κείμενα με ασαφή ή επικαλυπτόμενα συμφραζόμενα**, και σε αρκετές περιπτώσεις η λανθασμένη πρόβλεψη είναι κοινή για όλα.

🔠 Ερώτημα Γ.3 – Μεταβολή της παραμέτρου `MAX_WORDS`

Σε αυτό το ερώτημα μεταβάλλεται η παράμετρος `MAX_WORDS` από **25 σε 50**, ώστε να επιτρέψουμε μεγαλύτερη πληροφορία ανά δείγμα κειμένου.

🎯 Σκοπός

Να διερευνηθεί πώς επηρεάζεται:

- Η **απόδοση των μοντέλων** (accuracy)
- Ο **χρόνος εκπαίδευσης**
- Η **πολυπλοκότητα του input** (καθώς τα sequences μεγαλώνουν)

In [184…

```
MAX_WORDS = 50

results_50, preds_50 = run_experiments(model_configs)
```

```
🔄 Training: 1RNN
Epoch 1
100%|██████████| 118/118 [00:05<00:00, 23.46it/s]
Train Loss: 1.3765 | Val Loss: 1.3508 | Val Accuracy: 0.3312

Epoch 2
100%|██████████| 118/118 [00:04<00:00, 24.45it/s]
Train Loss: 1.3350 | Val Loss: 1.3147 | Val Accuracy: 0.3858

Epoch 3
100%|██████████| 118/118 [00:04<00:00, 24.34it/s]
Train Loss: 1.2998 | Val Loss: 1.3028 | Val Accuracy: 0.3853

Epoch 4
100%|██████████| 118/118 [00:04<00:00, 24.14it/s]
Train Loss: 1.3022 | Val Loss: 1.2859 | Val Accuracy: 0.4058

Epoch 5
100%|██████████| 118/118 [00:06<00:00, 19.14it/s]
Train Loss: 1.2738 | Val Loss: 1.2727 | Val Accuracy: 0.4218

Epoch 6
100%|██████████| 118/118 [00:05<00:00, 23.14it/s]
Train Loss: 1.2611 | Val Loss: 1.2595 | Val Accuracy: 0.4397

Epoch 7
100%|██████████| 118/118 [00:04<00:00, 25.26it/s]
Train Loss: 1.2739 | Val Loss: 1.2661 | Val Accuracy: 0.4355

Epoch 8
100%|██████████| 118/118 [00:04<00:00, 24.16it/s]
Train Loss: 1.2829 | Val Loss: 1.2809 | Val Accuracy: 0.4300

Epoch 9
100%|██████████| 118/118 [00:04<00:00, 24.88it/s]
Train Loss: 1.2629 | Val Loss: 1.2297 | Val Accuracy: 0.4874

Epoch 10
100%|██████████| 118/118 [00:04<00:00, 25.74it/s]
```

```
Train Loss: 1.2789 | Val Loss: 1.2759 | Val Accuracy: 0.4339

Epoch 11
100%|██████████| 118/118 [00:04<00:00, 25.69it/s]
Train Loss: 1.2644 | Val Loss: 1.2605 | Val Accuracy: 0.4328

Epoch 12
100%|██████████| 118/118 [00:04<00:00, 25.76it/s]
Train Loss: 1.2490 | Val Loss: 1.2511 | Val Accuracy: 0.4408

Epoch 13
100%|██████████| 118/118 [00:04<00:00, 25.29it/s]
Train Loss: 1.2091 | Val Loss: 1.1665 | Val Accuracy: 0.5518

Epoch 14
100%|██████████| 118/118 [00:05<00:00, 22.15it/s]
Train Loss: 1.2429 | Val Loss: 1.2772 | Val Accuracy: 0.4596

Epoch 15
100%|██████████| 118/118 [00:04<00:00, 23.94it/s]
Train Loss: 1.2618 | Val Loss: 1.3132 | Val Accuracy: 0.4203


🔄 Training: 1Bi-RNN
Epoch 1
100%|██████████| 118/118 [00:06<00:00, 18.36it/s]
Train Loss: 1.3781 | Val Loss: 1.3461 | Val Accuracy: 0.3393

Epoch 2
100%|██████████| 118/118 [00:06<00:00, 18.34it/s]
Train Loss: 1.3373 | Val Loss: 1.3376 | Val Accuracy: 0.3591

Epoch 3
100%|██████████| 118/118 [00:06<00:00, 18.37it/s]
Train Loss: 1.3443 | Val Loss: 1.3420 | Val Accuracy: 0.3509

Epoch 4
100%|██████████| 118/118 [00:09<00:00, 12.48it/s]
Train Loss: 1.2878 | Val Loss: 1.2828 | Val Accuracy: 0.3986

Epoch 5
100%|██████████| 118/118 [00:09<00:00, 12.20it/s]
Train Loss: 1.2909 | Val Loss: 1.3102 | Val Accuracy: 0.3680

Epoch 6
100%|██████████| 118/118 [00:09<00:00, 12.20it/s]
Train Loss: 1.2694 | Val Loss: 1.2621 | Val Accuracy: 0.4416

Epoch 7
100%|██████████| 118/118 [00:07<00:00, 15.14it/s]
Train Loss: 1.2814 | Val Loss: 1.2778 | Val Accuracy: 0.4038

Epoch 8
100%|██████████| 118/118 [00:06<00:00, 17.40it/s]
```

```
Train Loss: 1.2612 | Val Loss: 1.2576 | Val Accuracy: 0.4295

Epoch 9
100%|██████████| 118/118 [00:06<00:00, 17.44it/s]
Train Loss: 1.2428 | Val Loss: 1.2683 | Val Accuracy: 0.4238

Epoch 10
100%|██████████| 118/118 [00:07<00:00, 15.46it/s]
Train Loss: 1.2442 | Val Loss: 1.2363 | Val Accuracy: 0.4495

Epoch 11
100%|██████████| 118/118 [00:06<00:00, 17.09it/s]
Train Loss: 1.2286 | Val Loss: 1.2635 | Val Accuracy: 0.4246

Epoch 12
100%|██████████| 118/118 [00:06<00:00, 17.77it/s]
Train Loss: 1.2289 | Val Loss: 1.2349 | Val Accuracy: 0.4530

Epoch 13
100%|██████████| 118/118 [00:07<00:00, 15.89it/s]
Train Loss: 1.2236 | Val Loss: 1.2420 | Val Accuracy: 0.4464

Epoch 14
100%|██████████| 118/118 [00:09<00:00, 11.87it/s]
Train Loss: 1.2362 | Val Loss: 1.2527 | Val Accuracy: 0.4292

Epoch 15
100%|██████████| 118/118 [00:06<00:00, 16.95it/s]
Train Loss: 1.2194 | Val Loss: 1.2312 | Val Accuracy: 0.4503


🔄 Training: 2Bi-RNN
Epoch 1
100%|██████████| 118/118 [00:10<00:00, 10.84it/s]
Train Loss: 1.3570 | Val Loss: 1.3230 | Val Accuracy: 0.3995

Epoch 2
100%|██████████| 118/118 [00:11<00:00, 10.00it/s]
Train Loss: 1.3243 | Val Loss: 1.3657 | Val Accuracy: 0.2954

Epoch 3
100%|██████████| 118/118 [00:10<00:00, 10.81it/s]
Train Loss: 1.3132 | Val Loss: 1.2958 | Val Accuracy: 0.4029

Epoch 4
100%|██████████| 118/118 [00:10<00:00, 10.86it/s]
Train Loss: 1.2746 | Val Loss: 1.2862 | Val Accuracy: 0.4105

Epoch 5
100%|██████████| 118/118 [00:11<00:00, 10.35it/s]
Train Loss: 1.2756 | Val Loss: 1.2771 | Val Accuracy: 0.4204

Epoch 6
100%|██████████| 118/118 [00:10<00:00, 10.91it/s]
```

```
Train Loss: 1.2560 | Val Loss: 1.2965 | Val Accuracy: 0.4067

Epoch 7
100%|██████████| 118/118 [00:12<00:00,  9.36it/s]
Train Loss: 1.2336 | Val Loss: 1.2454 | Val Accuracy: 0.4855

Epoch 8
100%|██████████| 118/118 [00:11<00:00, 10.72it/s]
Train Loss: 1.2386 | Val Loss: 1.2708 | Val Accuracy: 0.4676

Epoch 9
100%|██████████| 118/118 [00:10<00:00, 10.91it/s]
Train Loss: 1.2563 | Val Loss: 1.2393 | Val Accuracy: 0.4857

Epoch 10
100%|██████████| 118/118 [00:15<00:00,  7.42it/s]
Train Loss: 1.3468 | Val Loss: 1.3730 | Val Accuracy: 0.3154

Epoch 11
100%|██████████| 118/118 [00:15<00:00,  7.49it/s]
Train Loss: 1.3598 | Val Loss: 1.3566 | Val Accuracy: 0.3339

Epoch 12
100%|██████████| 118/118 [00:12<00:00,  9.24it/s]
Train Loss: 1.3031 | Val Loss: 1.3222 | Val Accuracy: 0.3782

Epoch 13
100%|██████████| 118/118 [00:10<00:00, 10.85it/s]
Train Loss: 1.2819 | Val Loss: 1.2739 | Val Accuracy: 0.4483

Epoch 14
100%|██████████| 118/118 [00:11<00:00, 10.37it/s]
Train Loss: 1.2575 | Val Loss: 1.2609 | Val Accuracy: 0.4637

Epoch 15
100%|██████████| 118/118 [00:11<00:00, 10.54it/s]
Train Loss: 1.2495 | Val Loss: 1.2567 | Val Accuracy: 0.4487


🔄 Training: 1LSTM
Epoch 1
100%|██████████| 118/118 [00:04<00:00, 24.75it/s]
Train Loss: 1.3209 | Val Loss: 1.1462 | Val Accuracy: 0.6049

Epoch 2
100%|██████████| 118/118 [00:04<00:00, 25.59it/s]
Train Loss: 1.0443 | Val Loss: 0.9873 | Val Accuracy: 0.7630

Epoch 3
100%|██████████| 118/118 [00:04<00:00, 27.11it/s]
Train Loss: 0.9544 | Val Loss: 0.9408 | Val Accuracy: 0.8004

Epoch 4
100%|██████████| 118/118 [00:04<00:00, 25.45it/s]
```

```
Train Loss: 0.9125 | Val Loss: 0.9155 | Val Accuracy: 0.8262

Epoch 5
100%|██████████| 118/118 [00:07<00:00, 16.31it/s]
Train Loss: 0.8979 | Val Loss: 0.9019 | Val Accuracy: 0.8395

Epoch 6
100%|██████████| 118/118 [00:08<00:00, 14.59it/s]
Train Loss: 0.8901 | Val Loss: 0.9045 | Val Accuracy: 0.8370

Epoch 7
100%|██████████| 118/118 [00:05<00:00, 20.05it/s]
Train Loss: 0.8742 | Val Loss: 0.8870 | Val Accuracy: 0.8539

Epoch 8
100%|██████████| 118/118 [00:04<00:00, 27.25it/s]
Train Loss: 0.8619 | Val Loss: 0.8704 | Val Accuracy: 0.8701

Epoch 9
100%|██████████| 118/118 [00:04<00:00, 27.20it/s]
Train Loss: 0.8624 | Val Loss: 0.8724 | Val Accuracy: 0.8676

Epoch 10
100%|██████████| 118/118 [00:04<00:00, 26.82it/s]
Train Loss: 0.8540 | Val Loss: 0.8744 | Val Accuracy: 0.8678

Epoch 11
100%|██████████| 118/118 [00:05<00:00, 20.63it/s]
Train Loss: 0.8474 | Val Loss: 0.8635 | Val Accuracy: 0.8780

Epoch 12
100%|██████████| 118/118 [00:08<00:00, 14.69it/s]
Train Loss: 0.8433 | Val Loss: 0.8619 | Val Accuracy: 0.8789

Epoch 13
100%|██████████| 118/118 [00:04<00:00, 27.13it/s]
Train Loss: 0.8402 | Val Loss: 0.8647 | Val Accuracy: 0.8758

Epoch 14
100%|██████████| 118/118 [00:04<00:00, 26.83it/s]
Train Loss: 0.8456 | Val Loss: 0.8656 | Val Accuracy: 0.8747

Epoch 15
100%|██████████| 118/118 [00:05<00:00, 23.59it/s]
Train Loss: 0.8427 | Val Loss: 0.8610 | Val Accuracy: 0.8801


🔄 Training: 1Bi-LSTM
Epoch 1
100%|██████████| 118/118 [00:08<00:00, 14.24it/s]
Train Loss: 1.3149 | Val Loss: 1.1649 | Val Accuracy: 0.5651

Epoch 2
100%|██████████| 118/118 [00:13<00:00,  8.55it/s]
```

```
Train Loss: 1.0847 | Val Loss: 1.0248 | Val Accuracy: 0.7233

Epoch 3
100%|██████████| 118/118 [00:13<00:00,  8.59it/s]
Train Loss: 0.9878 | Val Loss: 0.9648 | Val Accuracy: 0.7786

Epoch 4
100%|██████████| 118/118 [00:13<00:00,  8.52it/s]
Train Loss: 0.9363 | Val Loss: 0.9441 | Val Accuracy: 0.7987

Epoch 5
100%|██████████| 118/118 [00:09<00:00, 12.98it/s]
Train Loss: 0.9098 | Val Loss: 0.9160 | Val Accuracy: 0.8251

Epoch 6
100%|██████████| 118/118 [00:08<00:00, 13.43it/s]
Train Loss: 0.8916 | Val Loss: 0.9051 | Val Accuracy: 0.8378

Epoch 7
100%|██████████| 118/118 [00:08<00:00, 13.19it/s]
Train Loss: 0.8769 | Val Loss: 0.8946 | Val Accuracy: 0.8468

Epoch 8
100%|██████████| 118/118 [00:09<00:00, 12.45it/s]
Train Loss: 0.8655 | Val Loss: 0.8868 | Val Accuracy: 0.8550

Epoch 9
100%|██████████| 118/118 [00:07<00:00, 16.36it/s]
Train Loss: 0.8611 | Val Loss: 0.8833 | Val Accuracy: 0.8574

Epoch 10
100%|██████████| 118/118 [00:07<00:00, 15.98it/s]
Train Loss: 0.8619 | Val Loss: 0.8850 | Val Accuracy: 0.8575

Epoch 11
100%|██████████| 118/118 [00:12<00:00,  9.28it/s]
Train Loss: 0.8608 | Val Loss: 0.8820 | Val Accuracy: 0.8599

Epoch 12
100%|██████████| 118/118 [00:07<00:00, 16.25it/s]
Train Loss: 0.8536 | Val Loss: 0.8778 | Val Accuracy: 0.8637

Epoch 13
100%|██████████| 118/118 [00:07<00:00, 16.22it/s]
Train Loss: 0.8586 | Val Loss: 0.8765 | Val Accuracy: 0.8643

Epoch 14
100%|██████████| 118/118 [00:10<00:00, 11.61it/s]
Train Loss: 0.8444 | Val Loss: 0.8764 | Val Accuracy: 0.8649

Epoch 15
100%|██████████| 118/118 [00:13<00:00,  8.54it/s]
```

```
Train Loss: 0.8404 | Val Loss: 0.8676 | Val Accuracy: 0.8730


🔄 Training: 2Bi-LSTM
Epoch 1
100%|██████████| 118/118 [00:12<00:00,  9.49it/s]
Train Loss: 1.2969 | Val Loss: 1.1574 | Val Accuracy: 0.5792

Epoch 2
100%|██████████| 118/118 [00:13<00:00,  8.59it/s]
Train Loss: 1.1084 | Val Loss: 1.0865 | Val Accuracy: 0.6562

Epoch 3
100%|██████████| 118/118 [00:19<00:00,  6.05it/s]
Train Loss: 1.0314 | Val Loss: 0.9964 | Val Accuracy: 0.7454

Epoch 4
100%|██████████| 118/118 [00:19<00:00,  5.90it/s]
Train Loss: 0.9692 | Val Loss: 0.9531 | Val Accuracy: 0.7866

Epoch 5
100%|██████████| 118/118 [00:16<00:00,  7.28it/s]
Train Loss: 0.9283 | Val Loss: 0.9252 | Val Accuracy: 0.8141

Epoch 6
100%|██████████| 118/118 [00:12<00:00,  9.82it/s]
Train Loss: 0.9024 | Val Loss: 0.9067 | Val Accuracy: 0.8334

Epoch 7
100%|██████████| 118/118 [00:15<00:00,  7.48it/s]
Train Loss: 0.8851 | Val Loss: 0.8942 | Val Accuracy: 0.8467

Epoch 8
100%|██████████| 118/118 [00:17<00:00,  6.65it/s]
Train Loss: 0.8733 | Val Loss: 0.8890 | Val Accuracy: 0.8522

Epoch 9
100%|██████████| 118/118 [00:12<00:00,  9.67it/s]
Train Loss: 0.8641 | Val Loss: 0.8798 | Val Accuracy: 0.8605

Epoch 10
100%|██████████| 118/118 [00:14<00:00,  8.16it/s]
Train Loss: 0.8557 | Val Loss: 0.8723 | Val Accuracy: 0.8686

Epoch 11
100%|██████████| 118/118 [00:19<00:00,  6.16it/s]
Train Loss: 0.8506 | Val Loss: 0.8671 | Val Accuracy: 0.8734

Epoch 12
100%|██████████| 118/118 [00:19<00:00,  6.03it/s]
Train Loss: 0.8437 | Val Loss: 0.8655 | Val Accuracy: 0.8733

Epoch 13
100%|██████████| 118/118 [00:12<00:00,  9.54it/s]
```

```
Train Loss: 0.8401 | Val Loss: 0.8577 | Val Accuracy: 0.8821

Epoch 14
100%|██████████| 118/118 [00:14<00:00,  8.24it/s]
Train Loss: 0.8360 | Val Loss: 0.8612 | Val Accuracy: 0.8796

Epoch 15
100%|██████████| 118/118 [00:19<00:00,  6.03it/s]
Train Loss: 0.8347 | Val Loss: 0.8627 | Val Accuracy: 0.8779
```

In [185…

```python
# Transpose results to match the table format in the assignment
df_transposed_50 = results_50.set_index("Model").T

# Round accuracy to 2 decimals, format numbers
df_display_50 = pd.DataFrame({
    col: [
        f"{df_transposed_50.loc['Accuracy', col]:.2f}",
        f"{int(df_transposed_50.loc['Parameters', col]):,}",
        f"{df_transposed_50.loc['Time (sec)', col]:.2f}"
    ]
    for col in df_transposed_50.columns
}, index=["Accuracy", "Parameters", "Time cost"])

df_display_50
```

Out[185…

	1RNN	1Bi-RNN	2Bi-RNN	1LSTM	1Bi-LSTM	2Bi-LSTM
Accuracy	42.03	45.03	44.87	88.01	87.30	87.79
Parameters	2,136,284	2,147,164	2,171,996	2,168,156	2,210,908	2,310,236
Time cost	5.10	7.88	12.25	5.54	10.44	16.42

Ερώτημα Γ.3 – Συμπεράσματα

Αλλάξαμε την παράμετρο `MAX_WORDS` από 25 σε 50 και επανεκπαιδεύσαμε όλα τα μοντέλα.

Τα αποτελέσματα έδειξαν:

- **Τα accuracies των απλών RNN (1RNN, 1Bi-RNN, 2Bi-RNN) έπεσαν κατακόρυφα**, σχεδόν στο μισό ή και πολύ περισσότερο κατα περίπτωση!
    - Π.χ. το 1RNN έπεσε από ~87% σε μόλις ~42%.
- **Τα LSTM-based μοντέλα (1LSTM, 1Bi-LSTM, 2Bi-LSTM) παρέμειναν σταθερά σε υψηλό επίπεδο** (περίπου 87–88%).
- **Ο χρόνος εκπαίδευσης αυξήθηκε σημαντικά**:
    - Π.χ. το 2Bi-LSTM ανέβηκε από ~8 sec σε ~16 sec ανά epoch.
- **Ο αριθμός παραμέτρων παρέμεινε ίδιος** σε όλα τα μοντέλα:
    - Αυτό ήταν αναμενόμενο, αφού ο αριθμός παραμέτρων εξαρτάται από:
        - Το μέγεθος του λεξιλογίου

- Το μέγεθος των embeddings
- Το μέγεθος του hidden layer
- Και **όχι** από το μήκος των εισόδων (`MAX_WORDS`).

📌 Γενικό Συμπέρασμα:

- Η αύξηση του μήκους των κειμένων σε 50 tokens:
  - Δυσκόλεψε τα απλά RNN μοντέλα (πιθανόν λόγω προβλημάτων "vanishing gradient" και αδυναμίας να διαχειριστούν μεγαλύτερες ακολουθίες).
  - Δεν επηρέασε σημαντικά τα LSTM μοντέλα, τα οποία είναι σχεδιασμένα για μακρύτερες ακολουθίες.
- Υπάρχει ξεκάθαρη επιβεβαίωση ότι **τα LSTM μοντέλα είναι πιο ανθεκτικά σε μεγαλύτερα κείμενα** και **διαχειρίζονται καλύτερα το context**.

🧠 Ερώτημα Γ.4 – Εκπαίδευση με Προεκπαιδευμένα GloVe Embeddings

Σε αυτό το ερώτημα γίνεται χρήση **προεκπαιδευμένων διανυσμάτων λέξεων (GloVe)** στη θέση των randomly initialized embeddings που χρησιμοποιήθηκαν στα προηγούμενα ερωτήματα.

Φόρτωση GloVe Embeddings

Τα GloVe vectors φορτώνονται από το αρχείο `glove.6B.100d.txt`, το οποίο περιέχει 400.000 λέξεις με διαστάσεις 100.

In [186…

```
# Διαβάζουμε το αρχείο
def load_glove_embeddings(glove_path):
    embeddings_index = {}
    with open(glove_path, encoding='utf8') as f:
        for line in f:
            values = line.strip().split()
            word = values[0]
            vector = torch.tensor([float(x) for x in values[1:]], dtype=t
            embeddings_index[word] = vector
    return embeddings_index

# Παράδειγμα
glove_embeddings = load_glove_embeddings('glove.6B.100d.txt')
print(f"Loaded {len(glove_embeddings)} word vectors from GloVe.")
```

```
Loaded 400000 word vectors from GloVe.
```

Δημιουργία Pretrained Embedding Matrix

Για να εισαχθούν τα vectors στο μοντέλο, φτιάχνεται μια embedding matrix:

In [187…

```
def create_embedding_matrix(vocab, glove_embeddings, embedding_dim):
    matrix = torch.zeros(len(vocab), embedding_dim)
    for word, idx in vocab.get_stoi().items():
```

```
        if word in glove_embeddings:
            matrix[idx] = glove_embeddings[word]
        else:
            matrix[idx] = torch.randn(embedding_dim) * 0.01  # μικρό rand
    return matrix
```

In [188…

```
embedding_matrix = create_embedding_matrix(vocab, glove_embeddings, EMBED
```

🛠️ Σημείωση για την Υλοποίηση – Τεχνικές Προσαρμογές Ερωτήματος 4

Για να υποστηριχθεί η χρήση προεκπαιδευμένων GloVe embeddings, έγιναν οι παρακάτω τροποποιήσεις στον υπάρχοντα κώδικα:

- Οι αλλαγές εντοπίζονται **μέσα στις συναρτήσεις και τις κλάσεις** με **σαφή inline σχολιασμό** της μορφής:

  ```
  # προσθήκη ερωτήματος 4
  ```

Εκτέλεση Πειραμάτων με GloVe Embeddings

Τα μοντέλα τρέχουν με το embedding_matrix ως αρχικοποίηση

In [ ]:

```
MAX_WORDS = 25

# Εκπαίδευση με προ-εκπαιδευμένα GloVe embeddings
results_glove, preds_glove = run_experiments(model_configs, pretrained_em
```

```
🔄 Training: 1RNN
Epoch 1
100%|██████████| 118/118 [00:03<00:00, 30.60it/s]
Train Loss: 1.0526 | Val Loss: 0.9051 | Val Accuracy: 0.8411

Epoch 2
100%|██████████| 118/118 [00:03<00:00, 31.82it/s]
Train Loss: 0.8857 | Val Loss: 0.8724 | Val Accuracy: 0.8705

Epoch 3
100%|██████████| 118/118 [00:03<00:00, 31.62it/s]
Train Loss: 0.8729 | Val Loss: 0.8904 | Val Accuracy: 0.8530

Epoch 4
100%|██████████| 118/118 [00:03<00:00, 31.19it/s]
Train Loss: 0.8719 | Val Loss: 0.8813 | Val Accuracy: 0.8632

Epoch 5
100%|██████████| 118/118 [00:03<00:00, 30.53it/s]
Train Loss: 0.8731 | Val Loss: 0.8791 | Val Accuracy: 0.8620

Epoch 6
100%|██████████| 118/118 [00:03<00:00, 29.52it/s]
Train Loss: 0.8750 | Val Loss: 0.8761 | Val Accuracy: 0.8663

Epoch 7
```

```
100%|██████████| 118/118 [00:03<00:00, 31.22it/s]
Train Loss: 0.8606 | Val Loss: 0.8628 | Val Accuracy: 0.8789

Epoch 8
100%|██████████| 118/118 [00:03<00:00, 30.44it/s]
Train Loss: 0.8558 | Val Loss: 0.8723 | Val Accuracy: 0.8707

Epoch 9
100%|██████████| 118/118 [00:03<00:00, 31.49it/s]
Train Loss: 0.8501 | Val Loss: 0.8596 | Val Accuracy: 0.8818

Epoch 10
100%|██████████| 118/118 [00:03<00:00, 32.43it/s]
Train Loss: 0.8495 | Val Loss: 0.8644 | Val Accuracy: 0.8780

Epoch 11
100%|██████████| 118/118 [00:03<00:00, 31.74it/s]
Train Loss: 0.8921 | Val Loss: 0.9048 | Val Accuracy: 0.8387

Epoch 12
100%|██████████| 118/118 [00:03<00:00, 30.58it/s]
Train Loss: 0.8826 | Val Loss: 0.8896 | Val Accuracy: 0.8513

Epoch 13
100%|██████████| 118/118 [00:04<00:00, 27.21it/s]
Train Loss: 0.8577 | Val Loss: 0.8630 | Val Accuracy: 0.8789

Epoch 14
100%|██████████| 118/118 [00:04<00:00, 28.31it/s]
Train Loss: 0.8488 | Val Loss: 0.8634 | Val Accuracy: 0.8775

Epoch 15
100%|██████████| 118/118 [00:03<00:00, 32.33it/s]
Train Loss: 0.8532 | Val Loss: 0.8791 | Val Accuracy: 0.8633


🔄 Training: 1Bi-RNN
Epoch 1
100%|██████████| 118/118 [00:04<00:00, 23.72it/s]
Train Loss: 1.0482 | Val Loss: 0.8979 | Val Accuracy: 0.8536

Epoch 2
100%|██████████| 118/118 [00:04<00:00, 26.83it/s]
Train Loss: 0.8860 | Val Loss: 0.8850 | Val Accuracy: 0.8600

Epoch 3
100%|██████████| 118/118 [00:04<00:00, 26.49it/s]
Train Loss: 0.8712 | Val Loss: 0.8885 | Val Accuracy: 0.8562

Epoch 4
100%|██████████| 118/118 [00:04<00:00, 26.86it/s]
Train Loss: 0.8695 | Val Loss: 0.8707 | Val Accuracy: 0.8725

Epoch 5
100%|██████████| 118/118 [00:05<00:00, 22.80it/s]
```

```
Train Loss: 0.8604 | Val Loss: 0.8707 | Val Accuracy: 0.8709

Epoch 6
100%|██████████| 118/118 [00:04<00:00, 25.24it/s]
Train Loss: 0.8612 | Val Loss: 0.8672 | Val Accuracy: 0.8745

Epoch 7
100%|██████████| 118/118 [00:04<00:00, 27.41it/s]
Train Loss: 0.8560 | Val Loss: 0.8627 | Val Accuracy: 0.8789

Epoch 8
100%|██████████| 118/118 [00:04<00:00, 27.22it/s]
Train Loss: 0.8556 | Val Loss: 0.8752 | Val Accuracy: 0.8674

Epoch 9
100%|██████████| 118/118 [00:04<00:00, 26.74it/s]
Train Loss: 0.8482 | Val Loss: 0.8586 | Val Accuracy: 0.8832

Epoch 10
100%|██████████| 118/118 [00:05<00:00, 23.48it/s]
Train Loss: 0.8528 | Val Loss: 0.8646 | Val Accuracy: 0.8786

Epoch 11
100%|██████████| 118/118 [00:04<00:00, 25.20it/s]
Train Loss: 0.8526 | Val Loss: 0.8876 | Val Accuracy: 0.8532

Epoch 12
100%|██████████| 118/118 [00:04<00:00, 24.13it/s]
Train Loss: 0.8553 | Val Loss: 0.8979 | Val Accuracy: 0.8437

Epoch 13
100%|██████████| 118/118 [00:04<00:00, 26.57it/s]
Train Loss: 0.8468 | Val Loss: 0.8561 | Val Accuracy: 0.8849

Epoch 14
100%|██████████| 118/118 [00:04<00:00, 25.01it/s]
Train Loss: 0.8409 | Val Loss: 0.8543 | Val Accuracy: 0.8867

Epoch 15
100%|██████████| 118/118 [00:04<00:00, 27.17it/s]
Train Loss: 0.8449 | Val Loss: 0.8568 | Val Accuracy: 0.8850


🔄 Training: 2Bi-RNN
Epoch 1
100%|██████████| 118/118 [00:07<00:00, 16.33it/s]
Train Loss: 0.9818 | Val Loss: 0.8718 | Val Accuracy: 0.8700

Epoch 2
100%|██████████| 118/118 [00:06<00:00, 17.22it/s]
Train Loss: 0.8795 | Val Loss: 0.8761 | Val Accuracy: 0.8645

Epoch 3
100%|██████████| 118/118 [00:06<00:00, 18.27it/s]
```

```
Train Loss: 0.8601 | Val Loss: 0.8661 | Val Accuracy: 0.8753

Epoch 4
100%|██████████| 118/118 [00:06<00:00, 17.56it/s]
Train Loss: 0.8534 | Val Loss: 0.8657 | Val Accuracy: 0.8757

Epoch 5
100%|██████████| 118/118 [00:09<00:00, 12.88it/s]
Train Loss: 0.8489 | Val Loss: 0.8680 | Val Accuracy: 0.8741

Epoch 6
100%|██████████| 118/118 [00:08<00:00, 13.77it/s]
Train Loss: 0.8462 | Val Loss: 0.8530 | Val Accuracy: 0.8880

Epoch 7
100%|██████████| 118/118 [00:06<00:00, 17.70it/s]
Train Loss: 0.8461 | Val Loss: 0.8799 | Val Accuracy: 0.8625

Epoch 8
100%|██████████| 118/118 [00:06<00:00, 18.29it/s]
Train Loss: 0.8426 | Val Loss: 0.8545 | Val Accuracy: 0.8874

Epoch 9
100%|██████████| 118/118 [00:08<00:00, 14.72it/s]
Train Loss: 0.8421 | Val Loss: 0.8588 | Val Accuracy: 0.8838

Epoch 10
100%|██████████| 118/118 [00:09<00:00, 13.02it/s]
Train Loss: 0.8406 | Val Loss: 0.8601 | Val Accuracy: 0.8812

Epoch 11
100%|██████████| 118/118 [00:09<00:00, 13.02it/s]
Train Loss: 0.8326 | Val Loss: 0.8467 | Val Accuracy: 0.8963

Epoch 12
100%|██████████| 118/118 [00:09<00:00, 12.99it/s]
Train Loss: 0.8327 | Val Loss: 0.8436 | Val Accuracy: 0.8976

Epoch 13
100%|██████████| 118/118 [00:08<00:00, 13.95it/s]
Train Loss: 0.8314 | Val Loss: 0.8520 | Val Accuracy: 0.8905

Epoch 14
100%|██████████| 118/118 [00:06<00:00, 18.45it/s]
Train Loss: 0.8290 | Val Loss: 0.8499 | Val Accuracy: 0.8913

Epoch 15
100%|██████████| 118/118 [00:06<00:00, 18.44it/s]
Train Loss: 0.8289 | Val Loss: 0.8471 | Val Accuracy: 0.8949


🔄 Training: 1LSTM
Epoch 1
100%|██████████| 118/118 [00:04<00:00, 29.46it/s]
```

```
Train Loss: 1.0227 | Val Loss: 0.8759 | Val Accuracy: 0.8697

Epoch 2
100%|██████████| 118/118 [00:03<00:00, 30.12it/s]
Train Loss: 0.8569 | Val Loss: 0.8532 | Val Accuracy: 0.8901

Epoch 3
100%|██████████| 118/118 [00:03<00:00, 29.51it/s]
Train Loss: 0.8402 | Val Loss: 0.8447 | Val Accuracy: 0.8963

Epoch 4
100%|██████████| 118/118 [00:03<00:00, 29.81it/s]
Train Loss: 0.8305 | Val Loss: 0.8406 | Val Accuracy: 0.8991

Epoch 5
100%|██████████| 118/118 [00:03<00:00, 29.72it/s]
Train Loss: 0.8230 | Val Loss: 0.8423 | Val Accuracy: 0.8988

Epoch 6
100%|██████████| 118/118 [00:04<00:00, 27.48it/s]
Train Loss: 0.8184 | Val Loss: 0.8415 | Val Accuracy: 0.9005

Epoch 7
100%|██████████| 118/118 [00:03<00:00, 30.18it/s]
Train Loss: 0.8138 | Val Loss: 0.8347 | Val Accuracy: 0.9062

Epoch 8
100%|██████████| 118/118 [00:04<00:00, 28.67it/s]
Train Loss: 0.8110 | Val Loss: 0.8333 | Val Accuracy: 0.9078

Epoch 9
100%|██████████| 118/118 [00:03<00:00, 29.76it/s]
Train Loss: 0.8073 | Val Loss: 0.8349 | Val Accuracy: 0.9057

Epoch 10
100%|██████████| 118/118 [00:04<00:00, 29.23it/s]
Train Loss: 0.8062 | Val Loss: 0.8327 | Val Accuracy: 0.9083

Epoch 11
100%|██████████| 118/118 [00:04<00:00, 26.62it/s]
Train Loss: 0.8037 | Val Loss: 0.8339 | Val Accuracy: 0.9067

Epoch 12
100%|██████████| 118/118 [00:04<00:00, 29.03it/s]
Train Loss: 0.8017 | Val Loss: 0.8344 | Val Accuracy: 0.9066

Epoch 13
100%|██████████| 118/118 [00:04<00:00, 29.20it/s]
Train Loss: 0.8001 | Val Loss: 0.8341 | Val Accuracy: 0.9071

Epoch 14
100%|██████████| 118/118 [00:04<00:00, 27.23it/s]
Train Loss: 0.7993 | Val Loss: 0.8331 | Val Accuracy: 0.9076

Epoch 15
100%|██████████| 118/118 [00:03<00:00, 29.85it/s]
```

```
Train Loss: 0.7979 | Val Loss: 0.8347 | Val Accuracy: 0.9059


🔄 Training: 1Bi-LSTM
Epoch 1
100%|██████████| 118/118 [00:04<00:00, 26.84it/s]
Train Loss: 1.0325 | Val Loss: 0.8733 | Val Accuracy: 0.8737

Epoch 2
100%|██████████| 118/118 [00:04<00:00, 27.60it/s]
Train Loss: 0.8574 | Val Loss: 0.8564 | Val Accuracy: 0.8866

Epoch 3
100%|██████████| 118/118 [00:04<00:00, 27.41it/s]
Train Loss: 0.8415 | Val Loss: 0.8466 | Val Accuracy: 0.8963

Epoch 4
100%|██████████| 118/118 [00:04<00:00, 27.16it/s]
Train Loss: 0.8320 | Val Loss: 0.8408 | Val Accuracy: 0.9005

Epoch 5
100%|██████████| 118/118 [00:06<00:00, 18.69it/s]
Train Loss: 0.8242 | Val Loss: 0.8370 | Val Accuracy: 0.9028

Epoch 6
100%|██████████| 118/118 [00:05<00:00, 20.71it/s]
Train Loss: 0.8189 | Val Loss: 0.8383 | Val Accuracy: 0.9028

Epoch 7
100%|██████████| 118/118 [00:04<00:00, 27.02it/s]
Train Loss: 0.8156 | Val Loss: 0.8362 | Val Accuracy: 0.9045

Epoch 8
100%|██████████| 118/118 [00:04<00:00, 28.01it/s]
Train Loss: 0.8115 | Val Loss: 0.8367 | Val Accuracy: 0.9046

Epoch 9
100%|██████████| 118/118 [00:04<00:00, 25.41it/s]
Train Loss: 0.8096 | Val Loss: 0.8352 | Val Accuracy: 0.9066

Epoch 10
100%|██████████| 118/118 [00:05<00:00, 22.42it/s]
Train Loss: 0.8064 | Val Loss: 0.8338 | Val Accuracy: 0.9087

Epoch 11
100%|██████████| 118/118 [00:04<00:00, 27.76it/s]
Train Loss: 0.8043 | Val Loss: 0.8337 | Val Accuracy: 0.9068

Epoch 12
100%|██████████| 118/118 [00:04<00:00, 27.62it/s]
Train Loss: 0.8021 | Val Loss: 0.8338 | Val Accuracy: 0.9068

Epoch 13
100%|██████████| 118/118 [00:04<00:00, 26.31it/s]
```

```
Train Loss: 0.8015 | Val Loss: 0.8334 | Val Accuracy: 0.9084

Epoch 14
100%|██████████| 118/118 [00:04<00:00, 24.71it/s]
Train Loss: 0.7998 | Val Loss: 0.8378 | Val Accuracy: 0.9041

Epoch 15
100%|██████████| 118/118 [00:07<00:00, 16.84it/s]
Train Loss: 0.7996 | Val Loss: 0.8330 | Val Accuracy: 0.9091


🔄 Training: 2Bi-LSTM
Epoch 1
100%|██████████| 118/118 [00:06<00:00, 18.32it/s]
Train Loss: 0.9845 | Val Loss: 0.8640 | Val Accuracy: 0.8788

Epoch 2
100%|██████████| 118/118 [00:08<00:00, 13.40it/s]
Train Loss: 0.8566 | Val Loss: 0.8505 | Val Accuracy: 0.8901

Epoch 3
100%|██████████| 118/118 [00:09<00:00, 12.36it/s]
Train Loss: 0.8404 | Val Loss: 0.8459 | Val Accuracy: 0.8959

Epoch 4
100%|██████████| 118/118 [00:09<00:00, 11.84it/s]
Train Loss: 0.8323 | Val Loss: 0.8414 | Val Accuracy: 0.9001

Epoch 5
100%|██████████| 118/118 [00:09<00:00, 12.14it/s]
Train Loss: 0.8253 | Val Loss: 0.8415 | Val Accuracy: 0.8999

Epoch 6
100%|██████████| 118/118 [00:11<00:00, 10.19it/s]
Train Loss: 0.8203 | Val Loss: 0.8392 | Val Accuracy: 0.9020

Epoch 7
100%|██████████| 118/118 [00:09<00:00, 12.82it/s]
Train Loss: 0.8168 | Val Loss: 0.8367 | Val Accuracy: 0.9046

Epoch 8
100%|██████████| 118/118 [00:08<00:00, 13.28it/s]
Train Loss: 0.8127 | Val Loss: 0.8369 | Val Accuracy: 0.9042

Epoch 9
100%|██████████| 118/118 [00:09<00:00, 12.08it/s]
Train Loss: 0.8098 | Val Loss: 0.8351 | Val Accuracy: 0.9067

Epoch 10
100%|██████████| 118/118 [00:09<00:00, 12.39it/s]
Train Loss: 0.8084 | Val Loss: 0.8392 | Val Accuracy: 0.9026

Epoch 11
100%|██████████| 118/118 [00:08<00:00, 14.07it/s]
```

```
Train Loss: 0.8064 | Val Loss: 0.8349 | Val Accuracy: 0.9061

Epoch 12
100%|██████████| 118/118 [00:09<00:00, 12.39it/s]
Train Loss: 0.8044 | Val Loss: 0.8326 | Val Accuracy: 0.9079

Epoch 13
100%|██████████| 118/118 [00:08<00:00, 14.00it/s]
Train Loss: 0.8037 | Val Loss: 0.8347 | Val Accuracy: 0.9054

Epoch 14
100%|██████████| 118/118 [00:09<00:00, 12.65it/s]
Train Loss: 0.8017 | Val Loss: 0.8326 | Val Accuracy: 0.9097

Epoch 15
100%|██████████| 118/118 [00:08<00:00, 13.68it/s]
Train Loss: 0.8006 | Val Loss: 0.8333 | Val Accuracy: 0.9079
```

In [190…

```python
# Transpose results to match the table format in the assignment
df_transposed_glove_25 = results_glove.set_index("Model").T

# Round accuracy to 2 decimals, format numbers
df_display_glove_25 = pd.DataFrame({
    col: [
        f"{df_transposed_glove_25.loc['Accuracy', col]:.2f}",
        f"{int(df_transposed_glove_25.loc['Parameters', col]):,}",
        f"{df_transposed_glove_25.loc['Time (sec)', col]:.2f}"
    ]
    for col in df_transposed_glove_25.columns
}, index=["Accuracy", "Parameters", "Time cost"])

df_display_glove_25
```

Out[190…

	1RNN	1Bi-RNN	2Bi-RNN	1LSTM	1Bi-LSTM	2Bi-LSTM
Accuracy	86.33	88.50	89.49	90.59	90.91	90.79
Parameters	2,136,284	2,147,164	2,171,996	2,168,156	2,210,908	2,310,236
Time cost	4.02	4.80	7.86	4.24	5.03	9.56

Ερώτημα Γ.4 – Χρήση Προεκπαιδευμένων Word Embeddings (GloVe) με MAX_WORDS = 25

Μετά την αλλαγή του `MAX_WORDS` πίσω σε 25 (όπως ζητούσε η εκφώνηση) και την εκπαίδευση των μοντέλων με προ-εκπαιδευμένα embeddings (GloVe 6B, 100d), προκύπτουν τα εξής:

- **Βελτίωση της ακρίβειας** σε όλα τα μοντέλα, περίπου κατά **1–2%** σε σχέση με τα randomly initialized embeddings.
- **Μικρή μείωση του χρόνου εκπαίδευσης** σε όλα τα μοντέλα (~0.5–1 sec ανά epoch).

- Πιθανότατα λόγω καλύτερης αρχικοποίησης, που επιτρέπει ταχύτερη σύγκλιση.
- **Καμία διαφορά στον αριθμό παραμέτρων**.
  - Το μέγεθος του μοντέλου εξαρτάται από το μέγεθος του λεξιλογίου και των κρυφών επιπέδων, όχι από το είδος της αρχικοποίησης των embeddings.

📌 Γενικό συμπέρασμα:

> Η χρήση προεκπαιδευμένων GloVe embeddings οδηγεί σε καλύτερη αρχική αναπαράσταση των λέξεων, βελτιώνοντας ελαφρώς την τελική ακρίβεια των μοντέλων χωρίς σημαντικό computational overhead.

🧊 Ερώτημα Γ.5 – Χρήση Προεκπαιδευμένων GloVe Embeddings με `freeze=True`

Σε αυτό το ερώτημα δοκιμάζουμε το πείραμα του Ερωτήματος 4 **με παγωμένα (frozen) embeddings**, ώστε να **μην ενημερώνονται τα διανύσματα** κατά τη διάρκεια της εκπαίδευσης.

🧪 Εκτέλεση Πειράματος

Η μόνη αλλαγή στην κλήση της συνάρτησης `run_experiments()` είναι η προσθήκη του ορίσματος `freeze_embeddings=True`:

In [79]:
```
results_glove_frozen, preds_glove_frozen = run_experiments(model_configs,
```

```
🔄 Training: 1RNN
Epoch 1
100%|██████████| 118/118 [00:05<00:00, 22.18it/s]
Train Loss: 1.0680 | Val Loss: 0.9012 | Val Accuracy: 0.8471

Epoch 2
100%|██████████| 118/118 [00:03<00:00, 35.35it/s]
Train Loss: 0.9006 | Val Loss: 0.9069 | Val Accuracy: 0.8354

Epoch 3
100%|██████████| 118/118 [00:03<00:00, 35.16it/s]
Train Loss: 0.8909 | Val Loss: 0.8859 | Val Accuracy: 0.8562

Epoch 4
100%|██████████| 118/118 [00:03<00:00, 34.51it/s]
Train Loss: 0.8858 | Val Loss: 0.8815 | Val Accuracy: 0.8612

Epoch 5
100%|██████████| 118/118 [00:03<00:00, 35.66it/s]
Train Loss: 0.8859 | Val Loss: 0.8833 | Val Accuracy: 0.8597

Epoch 6
100%|██████████| 118/118 [00:03<00:00, 34.94it/s]
```

```
Train Loss: 0.8852 | Val Loss: 0.8818 | Val Accuracy: 0.8603

Epoch 7
100%|██████████| 118/118 [00:03<00:00, 32.22it/s]
Train Loss: 0.8834 | Val Loss: 0.8767 | Val Accuracy: 0.8649

Epoch 8
100%|██████████| 118/118 [00:03<00:00, 32.65it/s]
Train Loss: 0.8799 | Val Loss: 0.8789 | Val Accuracy: 0.8630

Epoch 9
100%|██████████| 118/118 [00:03<00:00, 35.37it/s]
Train Loss: 0.8821 | Val Loss: 0.8775 | Val Accuracy: 0.8639

Epoch 10
100%|██████████| 118/118 [00:03<00:00, 35.38it/s]
Train Loss: 0.8771 | Val Loss: 0.8828 | Val Accuracy: 0.8601

Epoch 11
100%|██████████| 118/118 [00:03<00:00, 35.58it/s]
Train Loss: 0.8765 | Val Loss: 0.8799 | Val Accuracy: 0.8638

Epoch 12
100%|██████████| 118/118 [00:03<00:00, 35.77it/s]
Train Loss: 0.8761 | Val Loss: 0.8765 | Val Accuracy: 0.8654

Epoch 13
100%|██████████| 118/118 [00:03<00:00, 32.87it/s]
Train Loss: 0.8857 | Val Loss: 0.8773 | Val Accuracy: 0.8646

Epoch 14
100%|██████████| 118/118 [00:03<00:00, 35.60it/s]
Train Loss: 0.8766 | Val Loss: 0.8823 | Val Accuracy: 0.8597

Epoch 15
100%|██████████| 118/118 [00:03<00:00, 32.27it/s]
Train Loss: 0.8859 | Val Loss: 0.8765 | Val Accuracy: 0.8645


🔄 Training: 1Bi-RNN
Epoch 1
100%|██████████| 118/118 [00:05<00:00, 22.71it/s]
Train Loss: 1.0896 | Val Loss: 0.9450 | Val Accuracy: 0.8109

Epoch 2
100%|██████████| 118/118 [00:05<00:00, 22.83it/s]
Train Loss: 0.9178 | Val Loss: 0.9045 | Val Accuracy: 0.8396

Epoch 3
100%|██████████| 118/118 [00:05<00:00, 22.38it/s]
Train Loss: 0.8996 | Val Loss: 0.8975 | Val Accuracy: 0.8472

Epoch 4
100%|██████████| 118/118 [00:05<00:00, 22.51it/s]
```

```
Train Loss: 0.9083 | Val Loss: 0.8997 | Val Accuracy: 0.8418

Epoch 5
100%|██████████| 118/118 [00:05<00:00, 22.61it/s]
Train Loss: 0.8979 | Val Loss: 0.8915 | Val Accuracy: 0.8521

Epoch 6
100%|██████████| 118/118 [00:05<00:00, 21.14it/s]
Train Loss: 0.8922 | Val Loss: 0.9059 | Val Accuracy: 0.8371

Epoch 7
100%|██████████| 118/118 [00:05<00:00, 21.69it/s]
Train Loss: 0.8901 | Val Loss: 0.8903 | Val Accuracy: 0.8534

Epoch 8
100%|██████████| 118/118 [00:05<00:00, 22.83it/s]
Train Loss: 0.9151 | Val Loss: 0.8914 | Val Accuracy: 0.8509

Epoch 9
100%|██████████| 118/118 [00:05<00:00, 21.41it/s]
Train Loss: 0.8846 | Val Loss: 0.8808 | Val Accuracy: 0.8617

Epoch 10
100%|██████████| 118/118 [00:05<00:00, 22.61it/s]
Train Loss: 0.8845 | Val Loss: 0.8816 | Val Accuracy: 0.8603

Epoch 11
100%|██████████| 118/118 [00:05<00:00, 22.42it/s]
Train Loss: 0.8961 | Val Loss: 0.8831 | Val Accuracy: 0.8593

Epoch 12
100%|██████████| 118/118 [00:05<00:00, 21.90it/s]
Train Loss: 0.8803 | Val Loss: 0.8751 | Val Accuracy: 0.8661

Epoch 13
100%|██████████| 118/118 [00:05<00:00, 19.77it/s]
Train Loss: 0.8915 | Val Loss: 0.8928 | Val Accuracy: 0.8497

Epoch 14
100%|██████████| 118/118 [00:05<00:00, 21.99it/s]
Train Loss: 0.8967 | Val Loss: 0.8890 | Val Accuracy: 0.8526

Epoch 15
100%|██████████| 118/118 [00:05<00:00, 22.46it/s]
Train Loss: 0.8896 | Val Loss: 0.8891 | Val Accuracy: 0.8518


🔄 Training: 2Bi-RNN
Epoch 1
100%|██████████| 118/118 [00:08<00:00, 13.11it/s]
Train Loss: 1.0156 | Val Loss: 0.9186 | Val Accuracy: 0.8229

Epoch 2
100%|██████████| 118/118 [00:08<00:00, 13.33it/s]
```

```
Train Loss: 0.8944 | Val Loss: 0.8864 | Val Accuracy: 0.8547

Epoch 3
100%|██████████| 118/118 [00:09<00:00, 12.98it/s]
Train Loss: 0.8857 | Val Loss: 0.8911 | Val Accuracy: 0.8521

Epoch 4
100%|██████████| 118/118 [00:08<00:00, 13.13it/s]
Train Loss: 0.8919 | Val Loss: 0.8827 | Val Accuracy: 0.8582

Epoch 5
100%|██████████| 118/118 [00:08<00:00, 13.43it/s]
Train Loss: 0.8838 | Val Loss: 0.8848 | Val Accuracy: 0.8566

Epoch 6
100%|██████████| 118/118 [00:08<00:00, 13.63it/s]
Train Loss: 0.8834 | Val Loss: 0.8800 | Val Accuracy: 0.8614

Epoch 7
100%|██████████| 118/118 [00:08<00:00, 13.65it/s]
Train Loss: 0.8820 | Val Loss: 0.8793 | Val Accuracy: 0.8616

Epoch 8
100%|██████████| 118/118 [00:08<00:00, 13.13it/s]
Train Loss: 0.9006 | Val Loss: 0.9015 | Val Accuracy: 0.8396

Epoch 9
100%|██████████| 118/118 [00:08<00:00, 13.70it/s]
Train Loss: 0.8803 | Val Loss: 0.8887 | Val Accuracy: 0.8551

Epoch 10
100%|██████████| 118/118 [00:08<00:00, 13.80it/s]
Train Loss: 0.8836 | Val Loss: 0.8800 | Val Accuracy: 0.8625

Epoch 11
100%|██████████| 118/118 [00:08<00:00, 13.63it/s]
Train Loss: 0.8810 | Val Loss: 0.8785 | Val Accuracy: 0.8638

Epoch 12
100%|██████████| 118/118 [00:08<00:00, 13.86it/s]
Train Loss: 0.8804 | Val Loss: 0.8809 | Val Accuracy: 0.8600

Epoch 13
100%|██████████| 118/118 [00:08<00:00, 14.01it/s]
Train Loss: 0.8838 | Val Loss: 0.8838 | Val Accuracy: 0.8591

Epoch 14
100%|██████████| 118/118 [00:08<00:00, 13.38it/s]
Train Loss: 0.8919 | Val Loss: 0.8907 | Val Accuracy: 0.8511

Epoch 15
100%|██████████| 118/118 [00:08<00:00, 13.64it/s]
```

```
Train Loss: 0.9047 | Val Loss: 0.8913 | Val Accuracy: 0.8492


🔄 Training: 1LSTM
Epoch 1
100%|██████████| 118/118 [00:06<00:00, 17.68it/s]
Train Loss: 1.0298 | Val Loss: 0.8855 | Val Accuracy: 0.8616

Epoch 2
100%|██████████| 118/118 [00:06<00:00, 17.21it/s]
Train Loss: 0.8760 | Val Loss: 0.8706 | Val Accuracy: 0.8718

Epoch 3
100%|██████████| 118/118 [00:06<00:00, 18.07it/s]
Train Loss: 0.8647 | Val Loss: 0.8685 | Val Accuracy: 0.8741

Epoch 4
100%|██████████| 118/118 [00:06<00:00, 17.98it/s]
Train Loss: 0.8586 | Val Loss: 0.8605 | Val Accuracy: 0.8808

Epoch 5
100%|██████████| 118/118 [00:06<00:00, 17.63it/s]
Train Loss: 0.8543 | Val Loss: 0.8557 | Val Accuracy: 0.8855

Epoch 6
100%|██████████| 118/118 [00:06<00:00, 17.81it/s]
Train Loss: 0.8507 | Val Loss: 0.8527 | Val Accuracy: 0.8867

Epoch 7
100%|██████████| 118/118 [00:06<00:00, 18.01it/s]
Train Loss: 0.8484 | Val Loss: 0.8525 | Val Accuracy: 0.8886

Epoch 8
100%|██████████| 118/118 [00:07<00:00, 16.82it/s]
Train Loss: 0.8465 | Val Loss: 0.8511 | Val Accuracy: 0.8904

Epoch 9
100%|██████████| 118/118 [00:06<00:00, 17.21it/s]
Train Loss: 0.8435 | Val Loss: 0.8478 | Val Accuracy: 0.8943

Epoch 10
100%|██████████| 118/118 [00:06<00:00, 18.02it/s]
Train Loss: 0.8412 | Val Loss: 0.8526 | Val Accuracy: 0.8892

Epoch 11
100%|██████████| 118/118 [00:06<00:00, 18.18it/s]
Train Loss: 0.8393 | Val Loss: 0.8513 | Val Accuracy: 0.8893

Epoch 12
100%|██████████| 118/118 [00:06<00:00, 17.76it/s]
Train Loss: 0.8372 | Val Loss: 0.8487 | Val Accuracy: 0.8921

Epoch 13
100%|██████████| 118/118 [00:07<00:00, 16.78it/s]
```

```
Train Loss: 0.8373 | Val Loss: 0.8444 | Val Accuracy: 0.8949

Epoch 14
100%|██████████| 118/118 [00:06<00:00, 18.11it/s]
Train Loss: 0.8348 | Val Loss: 0.8452 | Val Accuracy: 0.8961

Epoch 15
100%|██████████| 118/118 [00:06<00:00, 17.65it/s]
Train Loss: 0.8327 | Val Loss: 0.8436 | Val Accuracy: 0.8976


🔄 Training: 1Bi-LSTM
Epoch 1
100%|██████████| 118/118 [00:12<00:00,  9.77it/s]
Train Loss: 1.0464 | Val Loss: 0.8823 | Val Accuracy: 0.8661

Epoch 2
100%|██████████| 118/118 [00:11<00:00, 10.33it/s]
Train Loss: 0.8750 | Val Loss: 0.8689 | Val Accuracy: 0.8743

Epoch 3
100%|██████████| 118/118 [00:11<00:00, 10.39it/s]
Train Loss: 0.8648 | Val Loss: 0.8624 | Val Accuracy: 0.8791

Epoch 4
100%|██████████| 118/118 [00:11<00:00, 10.35it/s]
Train Loss: 0.8594 | Val Loss: 0.8596 | Val Accuracy: 0.8822

Epoch 5
100%|██████████| 118/118 [00:11<00:00, 10.37it/s]
Train Loss: 0.8543 | Val Loss: 0.8554 | Val Accuracy: 0.8879

Epoch 6
100%|██████████| 118/118 [00:11<00:00, 10.01it/s]
Train Loss: 0.8510 | Val Loss: 0.8536 | Val Accuracy: 0.8883

Epoch 7
100%|██████████| 118/118 [00:11<00:00,  9.93it/s]
Train Loss: 0.8482 | Val Loss: 0.8499 | Val Accuracy: 0.8901

Epoch 8
100%|██████████| 118/118 [00:11<00:00, 10.28it/s]
Train Loss: 0.8455 | Val Loss: 0.8496 | Val Accuracy: 0.8912

Epoch 9
100%|██████████| 118/118 [00:11<00:00, 10.18it/s]
Train Loss: 0.8434 | Val Loss: 0.8512 | Val Accuracy: 0.8888

Epoch 10
100%|██████████| 118/118 [00:11<00:00, 10.28it/s]
Train Loss: 0.8417 | Val Loss: 0.8467 | Val Accuracy: 0.8943

Epoch 11
100%|██████████| 118/118 [00:11<00:00,  9.99it/s]
```

```
Train Loss: 0.8389 | Val Loss: 0.8451 | Val Accuracy: 0.8950

Epoch 12
100%|██████████| 118/118 [00:13<00:00,  9.01it/s]
Train Loss: 0.8379 | Val Loss: 0.8472 | Val Accuracy: 0.8933

Epoch 13
100%|██████████| 118/118 [00:14<00:00,  8.15it/s]
Train Loss: 0.8352 | Val Loss: 0.8458 | Val Accuracy: 0.8949

Epoch 14
100%|██████████| 118/118 [00:17<00:00,  6.74it/s]
Train Loss: 0.8345 | Val Loss: 0.8442 | Val Accuracy: 0.8967

Epoch 15
100%|██████████| 118/118 [00:15<00:00,  7.39it/s]
Train Loss: 0.8334 | Val Loss: 0.8424 | Val Accuracy: 0.8983


🔄 Training: 2Bi-LSTM
Epoch 1
100%|██████████| 118/118 [00:32<00:00,  3.65it/s]
Train Loss: 0.9900 | Val Loss: 0.8798 | Val Accuracy: 0.8613

Epoch 2
100%|██████████| 118/118 [00:28<00:00,  4.15it/s]
Train Loss: 0.8711 | Val Loss: 0.8709 | Val Accuracy: 0.8686

Epoch 3
100%|██████████| 118/118 [00:27<00:00,  4.37it/s]
Train Loss: 0.8635 | Val Loss: 0.8624 | Val Accuracy: 0.8797

Epoch 4
100%|██████████| 118/118 [00:29<00:00,  4.06it/s]
Train Loss: 0.8585 | Val Loss: 0.8589 | Val Accuracy: 0.8808

Epoch 5
100%|██████████| 118/118 [00:27<00:00,  4.30it/s]
Train Loss: 0.8547 | Val Loss: 0.8545 | Val Accuracy: 0.8867

Epoch 6
100%|██████████| 118/118 [00:25<00:00,  4.67it/s]
Train Loss: 0.8519 | Val Loss: 0.8520 | Val Accuracy: 0.8896

Epoch 7
100%|██████████| 118/118 [00:24<00:00,  4.80it/s]
Train Loss: 0.8492 | Val Loss: 0.8510 | Val Accuracy: 0.8899

Epoch 8
100%|██████████| 118/118 [00:25<00:00,  4.65it/s]
Train Loss: 0.8470 | Val Loss: 0.8494 | Val Accuracy: 0.8913

Epoch 9
100%|██████████| 118/118 [00:25<00:00,  4.60it/s]
```

```
Train Loss: 0.8455 | Val Loss: 0.8471 | Val Accuracy: 0.8943

Epoch 10
100%|██████████| 118/118 [00:25<00:00,  4.59it/s]
Train Loss: 0.8426 | Val Loss: 0.8486 | Val Accuracy: 0.8914

Epoch 11
100%|██████████| 118/118 [00:25<00:00,  4.60it/s]
Train Loss: 0.8396 | Val Loss: 0.8463 | Val Accuracy: 0.8953

Epoch 12
100%|██████████| 118/118 [00:26<00:00,  4.49it/s]
Train Loss: 0.8382 | Val Loss: 0.8443 | Val Accuracy: 0.8975

Epoch 13
100%|██████████| 118/118 [00:26<00:00,  4.43it/s]
Train Loss: 0.8373 | Val Loss: 0.8437 | Val Accuracy: 0.8972

Epoch 14
100%|██████████| 118/118 [00:26<00:00,  4.49it/s]
Train Loss: 0.8355 | Val Loss: 0.8428 | Val Accuracy: 0.8987

Epoch 15
100%|██████████| 118/118 [00:27<00:00,  4.32it/s]
Train Loss: 0.8337 | Val Loss: 0.8474 | Val Accuracy: 0.8934
```

In [80]:

```python
# Transpose results to match the table format in the assignment
df_transposed_glove_25_freeze = results_glove_frozen.set_index("Model").T

# Round accuracy to 2 decimals, format numbers
df_display_glove_25_freeze = pd.DataFrame({
    col: [
        f"{df_transposed_glove_25_freeze.loc['Accuracy', col]:.2f}",
        f"{int(df_transposed_glove_25_freeze.loc['Parameters', col]):,}",
        f"{df_transposed_glove_25_freeze.loc['Time (sec)', col]:.2f}"
    ]
    for col in df_transposed_glove_25_freeze.columns
}, index=["Accuracy", "Parameters", "Time cost"])

df_display_glove_25_freeze
```

Out[80]:

	1RNN	1Bi-RNN	2Bi-RNN	1LSTM	1Bi-LSTM	2Bi-LSTM
Accuracy	86.45	85.18	84.92	89.76	89.83	89.34
Parameters	10,884	21,764	46,596	42,756	85,508	184,836
Time cost	3.72	5.58	9.10	6.96	13.03	27.73

Ερώτημα Γ.5 – Συμπεράσματα

Στο πέμπτο ερώτημα, χρησιμοποιήθηκαν και πάλι τα GloVe προεκπαιδευμένα embeddings (`glove.6B.100d`), αλλά αυτή τη φορά το embedding layer παρέμεινε **"frozen"** (χωρίς ενημέρωση κατά τη διάρκεια της εκπαίδευσης).

Τα βασικά συμπεράσματα είναι:

- **Μικρή μείωση της ακρίβειας** σε όλα τα μοντέλα (~1%) σε σχέση με το σενάριο όπου τα embeddings ήταν trainable.
  - Ειδικά στα απλά RNN-based μοντέλα, η πτώση ήταν πιο αισθητή.
- **Μικρή βελτίωση του χρόνου εκπαίδευσης** (~0.5–1 sec ανά epoch), πιθανότατα επειδή δεν υπολογίζονταν gradients για το embedding layer.
- **Σημαντική μείωση στον αριθμό παραμέτρων**:
  - Το frozen embedding layer δεν έχει trainable parameters πλέον, οπότε οι συνολικοί παράμετροι του μοντέλου μειώθηκαν δραματικά.
  - Π.χ. `1RNN` είχε μόνο ~10.884 παραμέτρους με frozen embeddings, αντί για ~2 εκατομμύρια όταν ήταν trainable.

📌 Γενικό Συμπέρασμα:

> Το "πάγωμα" των embeddings μειώνει την πολυπλοκότητα και το χρόνο εκπαίδευσης, αλλά μπορεί να οδηγήσει σε μικρή πτώση της τελικής ακρίβειας. Το αποτέλεσμα εξαρτάται από το μοντέλο και το πόσο κατάλληλα είναι τα προεκπαιδευμένα embeddings για το συγκεκριμένο task.

🎬 Ερώτημα Γ.6 – Εφαρμογή σε νέο Dataset: IMDB Movie Reviews

Σε αυτό το ερώτημα εφαρμόζεται η ίδια διαδικασία με τα προηγούμενα, αλλά σε ένα **νέο dataset**: το IMDB dataset που περιλαμβάνει 50.000 κριτικές ταινιών, ταξινομημένες ως **θετικές** ή **αρνητικές**.

Εισαγωγή και Προεπεξεργασία Δεδομένων

- Το dataset έχει ληφθεί από Kaggle και περιέχει δύο στήλες:
  - `review` → το κείμενο της κριτικής
  - `sentiment` → η ετικέτα (`positive` , `negative`)
- Οι ετικέτες μετατράπηκαν σε αριθμούς: `positive = 1` , `negative = 0`
- Το dataset χωρίστηκε σε **training (80%)** και **test (20%)** με `train_test_split` και `stratify` ώστε να διατηρηθεί η ισορροπία των κατηγοριών.

In [191…

```
# Φόρτωση του IMDB dataset
dataset_imdb = pd.read_csv("IMDB Dataset.csv")

# Mapping από "positive"/"negative" σε 1/0
dataset_imdb["label"] = dataset_imdb["sentiment"].map({"positive": 1, "ne
```

```
# Split 80/20
from sklearn.model_selection import train_test_split

X_train_imdb, X_test_imdb, y_train_imdb, y_test_imdb = train_test_split(
    dataset_imdb["review"],
    dataset_imdb["label"],
    test_size=0.2,
    random_state=42,
    stratify=dataset_imdb["label"]
)
```

In [195…

```
# Δημιουργία dataset λιστών
train_dataset_imdb = [(label, text) for label, text in zip(y_train_imdb,
test_dataset_imdb = [(label, text) for label, text in zip(y_test_imdb, X_
```

Εκτέλεση Πειραμάτων

- Χρησιμοποιήθηκε **ξεχωριστό λεξιλόγιο** (`vocab_imdb`) που δημιουργήθηκε από τα train/test κείμενα του IMDB.
- Οι προβλέψεις έγιναν με τα 6 προκαθορισμένα μοντέλα (RNN / LSTM, 1-layer / 2-layer, uni/bidirectional).
- Το collate function προσαρμόστηκε με `shift_labels=False` καθώς οι ετικέτες είναι ήδη 0 και 1.

In [196…

```
# Χτίσιμο vocab_imdb IMDB
vocab_imdb = build_vocab_from_iterator(
    build_vocabulary([train_dataset_imdb, test_dataset_imdb]),
    min_freq=10,
    specials=["<PAD>", "<UNK>"]
)
vocab_imdb.set_default_index(vocab_imdb["<UNK>"])

print(f"Vocabulary size (IMDB): {len(vocab_imdb)}")
```

```
Vocabulary size (IMDB): 29065
```

In [197…

```
from torch.utils.data import DataLoader

train_loader_imdb = DataLoader(train_dataset_imdb, batch_size=BATCH_SIZE,
test_loader_imdb = DataLoader(test_dataset_imdb, batch_size=BATCH_SIZE, s
```

In [198…

```
df_imdb_results, preds_imdb = run_experiments(
    configs=model_configs,
    pretrained_embeddings=None,
    freeze_embeddings=False,
    vocab_to_use=vocab_imdb,
    output_dim=2,
    train_loader=train_loader_imdb,
    test_loader=test_loader_imdb
)
```

```
🔄 Training: 1RNN
Epoch 1
100%|██████████| 40/40 [00:03<00:00, 10.61it/s]
```

```
Train Loss: 0.6943 | Val Loss: 0.6918 | Val Accuracy: 0.5215

Epoch 2
100%|██████████| 40/40 [00:03<00:00, 13.29it/s]
Train Loss: 0.6844 | Val Loss: 0.6728 | Val Accuracy: 0.5850

Epoch 3
100%|██████████| 40/40 [00:02<00:00, 13.35it/s]
Train Loss: 0.6599 | Val Loss: 0.6537 | Val Accuracy: 0.6211

Epoch 4
100%|██████████| 40/40 [00:03<00:00, 13.28it/s]
Train Loss: 0.6348 | Val Loss: 0.6368 | Val Accuracy: 0.6516

Epoch 5
100%|██████████| 40/40 [00:03<00:00, 12.93it/s]
Train Loss: 0.6178 | Val Loss: 0.6258 | Val Accuracy: 0.6639

Epoch 6
100%|██████████| 40/40 [00:03<00:00, 13.22it/s]
Train Loss: 0.5977 | Val Loss: 0.6225 | Val Accuracy: 0.6687

Epoch 7
100%|██████████| 40/40 [00:03<00:00, 12.95it/s]
Train Loss: 0.5786 | Val Loss: 0.6128 | Val Accuracy: 0.6801

Epoch 8
100%|██████████| 40/40 [00:03<00:00, 12.30it/s]
Train Loss: 0.5701 | Val Loss: 0.6124 | Val Accuracy: 0.6823

Epoch 9
100%|██████████| 40/40 [00:03<00:00, 13.14it/s]
Train Loss: 0.5562 | Val Loss: 0.6066 | Val Accuracy: 0.6908

Epoch 10
100%|██████████| 40/40 [00:03<00:00, 13.09it/s]
Train Loss: 0.5435 | Val Loss: 0.6059 | Val Accuracy: 0.6887

Epoch 11
100%|██████████| 40/40 [00:03<00:00, 12.71it/s]
Train Loss: 0.5332 | Val Loss: 0.6023 | Val Accuracy: 0.6949

Epoch 12
100%|██████████| 40/40 [00:03<00:00, 13.10it/s]
Train Loss: 0.5288 | Val Loss: 0.6058 | Val Accuracy: 0.6885

Epoch 13
100%|██████████| 40/40 [00:03<00:00, 13.09it/s]
Train Loss: 0.5208 | Val Loss: 0.6033 | Val Accuracy: 0.6919

Epoch 14
100%|██████████| 40/40 [00:03<00:00, 13.12it/s]
Train Loss: 0.5090 | Val Loss: 0.5998 | Val Accuracy: 0.6996

Epoch 15
100%|██████████| 40/40 [00:03<00:00, 13.14it/s]
```

```
Train Loss: 0.5019 | Val Loss: 0.6039 | Val Accuracy: 0.6949


🔄 Training: 1Bi-RNN
Epoch 1
100%|██████████| 40/40 [00:03<00:00, 10.83it/s]
Train Loss: 0.6943 | Val Loss: 0.6919 | Val Accuracy: 0.5236

Epoch 2
100%|██████████| 40/40 [00:03<00:00, 11.84it/s]
Train Loss: 0.6888 | Val Loss: 0.6908 | Val Accuracy: 0.5250

Epoch 3
100%|██████████| 40/40 [00:03<00:00, 11.70it/s]
Train Loss: 0.6783 | Val Loss: 0.6764 | Val Accuracy: 0.5803

Epoch 4
100%|██████████| 40/40 [00:03<00:00, 11.69it/s]
Train Loss: 0.6530 | Val Loss: 0.6644 | Val Accuracy: 0.6117

Epoch 5
100%|██████████| 40/40 [00:03<00:00, 11.28it/s]
Train Loss: 0.6327 | Val Loss: 0.6402 | Val Accuracy: 0.6424

Epoch 6
100%|██████████| 40/40 [00:03<00:00, 11.70it/s]
Train Loss: 0.6099 | Val Loss: 0.6257 | Val Accuracy: 0.6620

Epoch 7
100%|██████████| 40/40 [00:04<00:00,  9.87it/s]
Train Loss: 0.5890 | Val Loss: 0.6160 | Val Accuracy: 0.6784

Epoch 8
100%|██████████| 40/40 [00:03<00:00, 10.93it/s]
Train Loss: 0.5727 | Val Loss: 0.6135 | Val Accuracy: 0.6823

Epoch 9
100%|██████████| 40/40 [00:03<00:00, 11.42it/s]
Train Loss: 0.5608 | Val Loss: 0.6079 | Val Accuracy: 0.6868

Epoch 10
100%|██████████| 40/40 [00:03<00:00, 11.47it/s]
Train Loss: 0.5442 | Val Loss: 0.6048 | Val Accuracy: 0.6933

Epoch 11
100%|██████████| 40/40 [00:03<00:00, 11.65it/s]
Train Loss: 0.5350 | Val Loss: 0.5990 | Val Accuracy: 0.6955

Epoch 12
100%|██████████| 40/40 [00:03<00:00, 11.81it/s]
Train Loss: 0.5255 | Val Loss: 0.6112 | Val Accuracy: 0.6816

Epoch 13
100%|██████████| 40/40 [00:03<00:00, 11.76it/s]
```

```
Train Loss: 0.5200 | Val Loss: 0.6005 | Val Accuracy: 0.6998

Epoch 14
100%|██████████| 40/40 [00:03<00:00, 11.54it/s]
Train Loss: 0.5086 | Val Loss: 0.5973 | Val Accuracy: 0.7023

Epoch 15
100%|██████████| 40/40 [00:03<00:00, 11.66it/s]
Train Loss: 0.5035 | Val Loss: 0.6003 | Val Accuracy: 0.6972


🔄 Training: 2Bi-RNN
Epoch 1
100%|██████████| 40/40 [00:04<00:00,  9.68it/s]
Train Loss: 0.6933 | Val Loss: 0.6923 | Val Accuracy: 0.5180

Epoch 2
100%|██████████| 40/40 [00:04<00:00,  9.67it/s]
Train Loss: 0.6806 | Val Loss: 0.6762 | Val Accuracy: 0.5808

Epoch 3
100%|██████████| 40/40 [00:04<00:00,  9.90it/s]
Train Loss: 0.6636 | Val Loss: 0.6683 | Val Accuracy: 0.6020

Epoch 4
100%|██████████| 40/40 [00:04<00:00,  9.88it/s]
Train Loss: 0.6417 | Val Loss: 0.6533 | Val Accuracy: 0.6214

Epoch 5
100%|██████████| 40/40 [00:04<00:00,  9.77it/s]
Train Loss: 0.6184 | Val Loss: 0.6314 | Val Accuracy: 0.6561

Epoch 6
100%|██████████| 40/40 [00:04<00:00,  8.84it/s]
Train Loss: 0.5980 | Val Loss: 0.6306 | Val Accuracy: 0.6570

Epoch 7
100%|██████████| 40/40 [00:04<00:00,  9.86it/s]
Train Loss: 0.5774 | Val Loss: 0.6117 | Val Accuracy: 0.6810

Epoch 8
100%|██████████| 40/40 [00:04<00:00,  9.60it/s]
Train Loss: 0.5623 | Val Loss: 0.6103 | Val Accuracy: 0.6811

Epoch 9
100%|██████████| 40/40 [00:04<00:00,  9.85it/s]
Train Loss: 0.5468 | Val Loss: 0.6043 | Val Accuracy: 0.6912

Epoch 10
100%|██████████| 40/40 [00:04<00:00,  9.73it/s]
Train Loss: 0.5353 | Val Loss: 0.6038 | Val Accuracy: 0.6937

Epoch 11
100%|██████████| 40/40 [00:04<00:00,  8.35it/s]
```

```
Train Loss: 0.5231 | Val Loss: 0.6050 | Val Accuracy: 0.6926

Epoch 12
100%|██████████| 40/40 [00:04<00:00,  9.54it/s]
Train Loss: 0.5145 | Val Loss: 0.6044 | Val Accuracy: 0.6932

Epoch 13
100%|██████████| 40/40 [00:04<00:00,  9.51it/s]
Train Loss: 0.5071 | Val Loss: 0.6030 | Val Accuracy: 0.6946

Epoch 14
100%|██████████| 40/40 [00:04<00:00,  9.72it/s]
Train Loss: 0.4948 | Val Loss: 0.6051 | Val Accuracy: 0.6920

Epoch 15
100%|██████████| 40/40 [00:04<00:00,  9.69it/s]
Train Loss: 0.4884 | Val Loss: 0.6028 | Val Accuracy: 0.6988


🔄 Training: 1LSTM
Epoch 1
100%|██████████| 40/40 [00:03<00:00, 11.82it/s]
Train Loss: 0.6922 | Val Loss: 0.6901 | Val Accuracy: 0.5354

Epoch 2
100%|██████████| 40/40 [00:03<00:00, 12.43it/s]
Train Loss: 0.6790 | Val Loss: 0.6631 | Val Accuracy: 0.6021

Epoch 3
100%|██████████| 40/40 [00:03<00:00, 12.28it/s]
Train Loss: 0.6351 | Val Loss: 0.6287 | Val Accuracy: 0.6521

Epoch 4
100%|██████████| 40/40 [00:03<00:00, 11.38it/s]
Train Loss: 0.5966 | Val Loss: 0.6061 | Val Accuracy: 0.6781

Epoch 5
100%|██████████| 40/40 [00:03<00:00, 12.00it/s]
Train Loss: 0.5716 | Val Loss: 0.6008 | Val Accuracy: 0.6823

Epoch 6
100%|██████████| 40/40 [00:03<00:00, 12.64it/s]
Train Loss: 0.5561 | Val Loss: 0.5940 | Val Accuracy: 0.6940

Epoch 7
100%|██████████| 40/40 [00:03<00:00, 12.41it/s]
Train Loss: 0.5406 | Val Loss: 0.5900 | Val Accuracy: 0.6980

Epoch 8
100%|██████████| 40/40 [00:03<00:00, 12.38it/s]
Train Loss: 0.5253 | Val Loss: 0.5900 | Val Accuracy: 0.7009

Epoch 9
100%|██████████| 40/40 [00:03<00:00, 12.18it/s]
```

```
Train Loss: 0.5128 | Val Loss: 0.5884 | Val Accuracy: 0.7034

Epoch 10
100%|██████████| 40/40 [00:03<00:00, 12.35it/s]
Train Loss: 0.5011 | Val Loss: 0.5891 | Val Accuracy: 0.7019

Epoch 11
100%|██████████| 40/40 [00:03<00:00, 12.42it/s]
Train Loss: 0.4901 | Val Loss: 0.5889 | Val Accuracy: 0.7068

Epoch 12
100%|██████████| 40/40 [00:03<00:00, 12.91it/s]
Train Loss: 0.4850 | Val Loss: 0.5988 | Val Accuracy: 0.6953

Epoch 13
100%|██████████| 40/40 [00:03<00:00, 12.59it/s]
Train Loss: 0.4720 | Val Loss: 0.5981 | Val Accuracy: 0.6968

Epoch 14
100%|██████████| 40/40 [00:03<00:00, 12.38it/s]
Train Loss: 0.4670 | Val Loss: 0.5952 | Val Accuracy: 0.7026

Epoch 15
100%|██████████| 40/40 [00:03<00:00, 12.86it/s]
Train Loss: 0.4567 | Val Loss: 0.5995 | Val Accuracy: 0.6990


🔄 Training: 1Bi-LSTM
Epoch 1
100%|██████████| 40/40 [00:03<00:00, 11.62it/s]
Train Loss: 0.6919 | Val Loss: 0.6900 | Val Accuracy: 0.5335

Epoch 2
100%|██████████| 40/40 [00:03<00:00, 11.96it/s]
Train Loss: 0.6768 | Val Loss: 0.6615 | Val Accuracy: 0.6074

Epoch 3
100%|██████████| 40/40 [00:03<00:00, 10.99it/s]
Train Loss: 0.6365 | Val Loss: 0.6278 | Val Accuracy: 0.6535

Epoch 4
100%|██████████| 40/40 [00:03<00:00, 11.58it/s]
Train Loss: 0.6007 | Val Loss: 0.6061 | Val Accuracy: 0.6791

Epoch 5
100%|██████████| 40/40 [00:03<00:00, 11.80it/s]
Train Loss: 0.5807 | Val Loss: 0.5985 | Val Accuracy: 0.6888

Epoch 6
100%|██████████| 40/40 [00:03<00:00, 11.90it/s]
Train Loss: 0.5604 | Val Loss: 0.5969 | Val Accuracy: 0.6892

Epoch 7
100%|██████████| 40/40 [00:03<00:00, 10.50it/s]
```

```
Train Loss: 0.5449 | Val Loss: 0.5871 | Val Accuracy: 0.7029

Epoch 8
100%|██████████| 40/40 [00:03<00:00, 10.67it/s]
Train Loss: 0.5307 | Val Loss: 0.5874 | Val Accuracy: 0.7049

Epoch 9
100%|██████████| 40/40 [00:03<00:00, 11.04it/s]
Train Loss: 0.5179 | Val Loss: 0.5884 | Val Accuracy: 0.7027

Epoch 10
100%|██████████| 40/40 [00:03<00:00, 11.63it/s]
Train Loss: 0.5081 | Val Loss: 0.5862 | Val Accuracy: 0.7048

Epoch 11
100%|██████████| 40/40 [00:03<00:00, 11.62it/s]
Train Loss: 0.4994 | Val Loss: 0.5897 | Val Accuracy: 0.7004

Epoch 12
100%|██████████| 40/40 [00:03<00:00, 11.63it/s]
Train Loss: 0.4887 | Val Loss: 0.5944 | Val Accuracy: 0.6973

Epoch 13
100%|██████████| 40/40 [00:03<00:00, 10.15it/s]
Train Loss: 0.4780 | Val Loss: 0.5897 | Val Accuracy: 0.7045

Epoch 14
100%|██████████| 40/40 [00:03<00:00, 11.29it/s]
Train Loss: 0.4708 | Val Loss: 0.5880 | Val Accuracy: 0.7074

Epoch 15
100%|██████████| 40/40 [00:03<00:00, 10.94it/s]
Train Loss: 0.4586 | Val Loss: 0.5935 | Val Accuracy: 0.7046


🔄 Training: 2Bi-LSTM
Epoch 1
100%|██████████| 40/40 [00:04<00:00,  9.99it/s]
Train Loss: 0.6864 | Val Loss: 0.6753 | Val Accuracy: 0.5768

Epoch 2
100%|██████████| 40/40 [00:04<00:00,  8.58it/s]
Train Loss: 0.6488 | Val Loss: 0.6379 | Val Accuracy: 0.6343

Epoch 3
100%|██████████| 40/40 [00:03<00:00, 10.00it/s]
Train Loss: 0.6090 | Val Loss: 0.6099 | Val Accuracy: 0.6759

Epoch 4
100%|██████████| 40/40 [00:04<00:00,  9.99it/s]
Train Loss: 0.5815 | Val Loss: 0.5999 | Val Accuracy: 0.6826

Epoch 5
100%|██████████| 40/40 [00:04<00:00,  9.98it/s]
```

```
Train Loss: 0.5575 | Val Loss: 0.5942 | Val Accuracy: 0.6944

Epoch 6
100%|██████████| 40/40 [00:04<00:00,  9.91it/s]
Train Loss: 0.5402 | Val Loss: 0.5970 | Val Accuracy: 0.6954

Epoch 7
100%|██████████| 40/40 [00:03<00:00, 10.08it/s]
Train Loss: 0.5237 | Val Loss: 0.5968 | Val Accuracy: 0.6939

Epoch 8
100%|██████████| 40/40 [00:03<00:00, 10.05it/s]
Train Loss: 0.5120 | Val Loss: 0.5969 | Val Accuracy: 0.6912

Epoch 9
100%|██████████| 40/40 [00:03<00:00, 10.16it/s]
Train Loss: 0.4993 | Val Loss: 0.5945 | Val Accuracy: 0.7022

Epoch 10
100%|██████████| 40/40 [00:04<00:00,  8.21it/s]
Train Loss: 0.4878 | Val Loss: 0.5910 | Val Accuracy: 0.7071

Epoch 11
100%|██████████| 40/40 [00:04<00:00,  9.87it/s]
Train Loss: 0.4751 | Val Loss: 0.5944 | Val Accuracy: 0.7067

Epoch 12
100%|██████████| 40/40 [00:03<00:00, 10.11it/s]
Train Loss: 0.4672 | Val Loss: 0.6014 | Val Accuracy: 0.6997

Epoch 13
100%|██████████| 40/40 [00:04<00:00,  9.80it/s]
Train Loss: 0.4565 | Val Loss: 0.5989 | Val Accuracy: 0.7028

Epoch 14
100%|██████████| 40/40 [00:04<00:00,  9.02it/s]
Train Loss: 0.4536 | Val Loss: 0.6015 | Val Accuracy: 0.7013

Epoch 15
100%|██████████| 40/40 [00:05<00:00,  7.28it/s]
Train Loss: 0.4497 | Val Loss: 0.6036 | Val Accuracy: 0.7004
```

In [206…

```python
# Transpose results to match the table format in the assignment
df_transposed_imdb = df_imdb_results.set_index("Model").T

# Round accuracy to 2 decimals, format numbers
df_display_imdb = pd.DataFrame({
    col: [
        f"{df_transposed_imdb.loc['Accuracy', col]:.2f}",
        f"{int(df_transposed_imdb.loc['Parameters', col]):,}",
        f"{df_transposed_imdb.loc['Time (sec)', col]:.2f}"
    ]
    for col in df_transposed_imdb.columns
}, index=["Accuracy", "Parameters", "Time cost"])

df_display_imdb
```

Out[206…

	1RNN	1Bi-RNN	2Bi-RNN	1LSTM	1Bi-LSTM	2Bi-LSTM
Accuracy	69.49	69.72	69.88	69.90	70.46	70.04
Parameters	2,917,254	2,928,006	2,952,838	2,949,126	2,991,750	3,091,078
Time cost	3.76	4.18	4.92	3.98	4.29	5.06

Ερώτημα Γ.6 – Συμπεράσματα

- Όλες οι προσεγγίσεις εμφάνισαν **παρόμοια ακρίβεια**, με ελαφρώς καλύτερες επιδόσεις τα **Bi-Directional μοντέλα**.
- Τα **LSTM** είχαν ελαφρώς καλύτερα αποτελέσματα από τα αντίστοιχα **RNN**, όπως ήταν αναμενόμενο λόγω της πιο εξελιγμένης αρχιτεκτονικής τους.
- Οι **χρόνοι εκπαίδευσης** αυξάνονται αισθητά με την πολυπλοκότητα του μοντέλου (ιδιαίτερα στο `2Bi-LSTM`).
- Αν και τα πιο σύνθετα μοντέλα προσφέρουν μικρή αύξηση στην ακρίβεια, **το απλό `1RNN`** φαίνεται να προσφέρει **καλή ισορροπία** μεταξύ:
  - Απόδοσης
  - Χρόνου εκπαίδευσης
  - Αριθμού παραμέτρων

📌 Συμπερασματικά, **το `1RNN` είναι πιθανώς η βέλτιστη επιλογή στην παρούσα φάση** όταν προτεραιότητα είναι η ταχύτητα και η απλότητα, χωρίς μεγάλη θυσία στην ακρίβεια.