Nama : Michael christopher
NIM : 1103210260
Tugas Week 4 - Analisa

1 Pendahuluan

Model rekuren (RNN, LSTM, GRU) masih menjadi alternatif ringan-parameter bagi transformer, khususnya pada skenario teks berdurasi pendek-menengah. Studi singkat ini menyelidiki performa ketiga arsitektur tersebut di bawah dua platform dominan—TensorFlow-Keras dan PyTorch—dengan protokol pelatihan yang sengaja dijaga identik kecuali fitur default masing-masing framework.

2 Metodologi

2.1 Dataset dan Pra-proses

Korpus IMDb berjumlah 50 000 ulasan berbahasa Inggris dipakai sebagai objek kajian. Data diacak ulang dan disegmentasi menjadi 20 000 contoh latih, 5 000 validasi, dan 25 000 uji, menjaga rasio kelas 1:1. Tokenisasi dibatasi pada 30 000 lemma teratas, diikuti padding/truncating hingga panjang maksimum 400 token. Setiap token diproyeksikan ke embedding berdimensi 128 (≈ 3,84 × 10^6 parameter).

2.2 Arsitektur

Masing-masing model terdiri atas dua lapis sel rekuren unidirectional berukuran 64 unit, diapit dropout 0,2 dan diakhiri dense-sigmoid. Rumus teoretik parameter inti $P=(d+h)h+h2+hP = (d + h)h + h^2 + hP=(d+h)h+h2+h$ memberi 12 352 parameter untuk RNN klasik, 49 408 untuk LSTM (×4), dan 37 056 untuk GRU (×3).

2.3 Protokol Pelatihan

Semua eksperimen dijalankan sepuluh epoch dengan Adam ($\alpha_0 = 10^{-3}$), batch 32, pada GPU CUDA 12.4 (VRAM 8 GB). Sisi Keras tetap pada hiper-parameter baku; sisi PyTorch menambahkan (i) inisialisasi orthogonal pada bobot rekuren, (ii) kliping norma gradien ke 5, dan (iii) scheduler **ReduceLROnPlateau**. DataLoader PyTorch menggunakan padding dinamis, sedangkan `pad_sequences` Keras menghasilkan tensor statik 400 × 128.

3 Hasil

Pada platform Keras, ketiga arsitektur hanya mencapai kinerja dekat acak: akurasi 0,500 (RNN), 0,511 (LSTM), dan 0,507 (GRU), dengan F1-score < 0,36. Pada platform PyTorch, RNN dan LSTM tetap stagnan (≈ 0,51), tetapi GRU mencatat lonjakan dramatis: akurasi 0,868, presisi 0,850, recall 0,893, F1 0,871, dan AUC 0,942. Waktu pelatihan rata-rata per epoch ialah 40 s (RNN), 80 s (LSTM), dan 60 s (GRU).

4 Diskusi

Vanishing gradient diamati menjadi penyebab utama kegagalan RNN-Keras; ketiadaan gerbang mempercepat peluruhan sinyal pada sekuens 400 langkah. LSTM secara teoritis memecahkan masalah tersebut, namun kompleksitas empat gerbang meningkatkan sensitivitas terhadap inisialisasi dan fluktuasi gradien: tanpa orthogonal init dan kliping

eksplisit, LSTM tersangkut di dataran loss awal. GRU memberikan kompromi struktur—tiga gerbang yang cukup untuk stabilitas, tetapi 25 % lebih ringan daripada LSTM—sehingga lebih mudah dikonvergensikan dengan protokol pelatihan ketat.

Keberhasilan GRU-PyTorch menegaskan peran *training protocol* setara dengan arsitektur. Orthogonal init menjaga spektrum eigen matriks rekuren sehingga norma gradien bertahan mendekati satu; kliping gradien menahan luapan sesaat yang jika tidak akan membuat Adam menurunkan αt\alpha_tαt secara drastis; penurun laju belajar adaptif memfasilitasi transisi keluar dari plateau; dan padding dinamis menurunkan panjang rata-rata langkah waktu hingga ± 280, yang berdampak pada kestabilan dan efisiensi GPU.

5 Keterbatasan dan Riset Lanjutan

Ukuran hidden hanya 64; kemungkinan LSTM melejit pada h≥128h \ge 128h≥128 belum diuji. Arsitektur bidirectional serta regularisasi tambahan (weight decay, layer norm) juga belum dieksplorasi. Dataset IMDb tergolong bersih; penerapan pada domain berderau tinggi (mis. Twitter) dapat menghasilkan dinamika berbeda.

6 Kesimpulan

Dalam kondisi standar, RNN dan LSTM gagal melewati akurasi tebakan acak di kedua framework. Hanya GRU yang sanggup memanfaatkan tuning agresif dan, di bawah PyTorch dengan inisialisasi orthogonal, kliping gradien, serta scheduler adaptif, mencapai akurasi 86,8 %—naik ±36 poin persentase dibanding seluruh konfigurasi lain. Oleh karena itu, untuk tugas analisis sentimen berskala menengah, GRU yang dilatih dengan protokol serupa direkomendasikan sebagai baseline praktis; pengguna TensorFlow-Keras perlu mereplikasi teknik kliping dan inisialisasi orthogonal guna memperoleh performa sepadan, sementara LSTM sebaiknya dipertimbangkan hanya ketika dependensi konteks sangat panjang atau kapasitas model sengaja diperbesar.

Diskusi (Ringkas)

- **RNN klasik** menunjukkan akurasi 50 % di kedua framework; absennya gerbang membuatnya rentan vanishing gradient pada sekuens 400 token.

- **LSTM** seharusnya memecahkan masalah ini, tetapi kompleksitas empat gerbang + inisialisasi Glorot-uniform (default Keras) + ketiadaan gradient-clipping menyebabkan konvergensinya mandek di 51 %.

- **GRU** memiliki kompleksitas lebih rendah daripada LSTM, tetap menjaga gating esensial, dan—jika dipadukan dengan inisialisasi orthogonal, gradient-clipping, dan scheduler laju-belajar adaptif—mampu mempertahankan sinyal gradien sehingga memperoleh **akurasi 86,8 % dan AUC 0,94**.

- Perbedaan terbesar antar-framework terletak pada **inisialisasi bobot rekuren** (orthogonal default di PyTorch, Glorot di Keras) serta **kemudahan set kliping gradien**; keduanya terbukti krusial bagi model rekuren.

- **SimpleRNN (Keras) – “negatif mutlakh”**
  RNN di TensorFlow mencapai akurasi 0,500 tetapi presisi, recall, dan F1-score kelas positif persis 0,00; AUC bertahan 0,503. Angka-angka ini mengindikasikan model menganut *strategy of one class*: ia memproduksi probabilitas sangat rendah untuk label 1 dan, akibatnya, menebak seluruh sampel sebagai negatif. Situasi ini menekan false-positive ke nol (presisi=0) namun mengangkat false-negative ke maksimum (recall=0). AUC yang sedikit di atas 0,5 murni konsekuensi numerik dari distribusi skor yang sempit di bawah ambang, bukan bukti pembeda pola.

- **SimpleRNN (PyTorch) – “positif massal”**
  Kebalikan dari kasus di atas, implementasi PyTorch memiliki presisi 0,501 dan recall 0,939 pada kelas positif, memproduksi 24 500 prediksi positif dari 25 000 sampel uji. Akurasi tetap 0,502—setengah prediksi positif ternyata salah—tetapi recall yang nyaris sempurna menaikkan F1 ke 0,653. Di ROC-space, keputusan yang terlalu permisif ini tercermin pada AUC 0,507, nyaris garis acak. Dengan kata lain, **kedua framework mengekspresikan bias kelas ekstrem yang berbeda**—negatif mutlak di Keras, positif massal di PyTorch—namun keduanya tidak mempelajari fitur bermakna.

- **LSTM (Keras)**
  LSTM menawarkan perbaikan numerik kecil: akurasi 0,511, presisi 0,519, recall 0,271, F1 0,356, AUC 0,513. Model ini menebak label positif jauh lebih jarang daripada jumlah sebenarnya (recall rendah) tetapi ketika ia menebak, probabilitasnya cukup selektif untuk menghasilkan presisi >0,5. Akurasi naik +1,1 pp (percentage-point) dibanding RNN, menandakan gate input/forget berhasil menahan sebagian degradasi sinyal, tetapi masih belum cukup untuk melampaui baseline 0,5 secara substansial.

- **LSTM (PyTorch)**
  Metriknya hampir identik di akurasi (0,511) namun memperlihatkan kontras presisi-recall: presisi 0,571 disertai recall 0,087 (F1 0,151). Pola ini menandakan model “sangat berhati-hati”: ia mengeluarkan prediksi positif pada sebagian kecil contoh (hanya ± 2 000 dari 25 000), namun setengahnya benar. Kebijakan internal tersebut—diduga implikasi kliping gradien—menghasilkan ROC yang masih hanya 0,519, sedikit di atas acak.

- **GRU (Keras)**
  Meskipun memiliki gating lebih ringkas serta teorinya lebih stabil, GRU-Keras tetap stagnan: akurasi 0,507, presisi 0,535, recall 0,110, F1 0,182, AUC 0,506. Artinya, tanpa perlakuan stabilisasi tambahan, GRU mengalami plateau yang sama dengan LSTM dan belum mampu memanfaatkan kapasitasnya.

- **GRU (PyTorch)**
  Di sinilah dampak protokol pelatihan agresif terlihat jelas. GRU-PyTorch membukukan **akurasi 0,868, presisi 0,850, recall 0,893, F1 0,871, dan AUC 0,942**. Nilai-nilai tinggi di seluruh metrik menunjukkan keseimbangan prediksi di kedua kelas: false-positive dan false-negative menurun drastis, kurva ROC melengkung tajam menuju sudut kiri-atas, dan F1—harmonik presisi-recall—nyaris setara dengan

akurasi, menandakan konsistensi.

Interpretasi Lintas-Model

- **Bias kelas ekstrem** pada SimpleRNN di kedua framework menyiratkan *signal-to-noise ratio* terlalu rendah; model memilih memaksimalkan akurasi trivial dengan cara memprediksi satu kelas.

- **LSTM** meningkatkan presisi berkat mekanisme “forget gate” yang memangkas rangkaian sinyal lemah, namun tanpa kliping + penjadwalan laju-belajar, gate tidak sempat dikalibrasi—hasilnya recall anjlok.

- **GRU** di TensorFlow gagal beranjak karena faktor-faktor sama yang merugikan LSTM, tetapi struktur tiga gerbangnya menjadi dambaan setelah orthogonal-init dan kliping diterapkan di PyTorch: kapasitas cukup besar untuk belajar pola, cukup ringan untuk cepat konvergen, dan cukup stabil untuk mempertahankan gradien.

- **AUC** menggarisbawahi kesimpulan di atas. Setiap model “gagal” memiliki AUC ~0,51 (garis acak), sedangkan GRU-PyTorch mencatat 0,94—gap 0,43 di skala yang mengukur kemampuan ranking antar kelas.