PENGEMBANGAN MESIN CATUR MENGGUNAKAN LONG SHORT TERM MEMORY (LSTM)

SKRIPSI

Disusun oleh:

Rafi Indra Fattah

NIM: 205150200111006



PROGRAM STUDI TEKNIK INFORMATIKA

JURUSAN TEKNIK INFORMATIKA

FAKULTAS ILMU KOMPUTER

UNIVERSITAS BRAWIJAYA

MALANG

2023

DAFTAR ISI

[DAFTAR ISI ii](#_Toc152698115)

[DAFTAR TABEL v](#_Toc152698116)

[DAFTAR GAMBAR vi](#_Toc152698117)

[BAB 1 PENDAHULUAN 1](#_Toc152698118)

[1.1 Latar Belakang 1](#_Toc152698119)

[1.2 Rumusan Masalah 2](#_Toc152698120)

[1.3 Tujuan 2](#_Toc152698121)

[1.4 Manfaat 2](#_Toc152698122)

[1.5 Batasan Masalah 2](#_Toc152698123)

[1.6 Sistematika Pembahasan 3](#_Toc152698124)

[BAB 2 LANDASAN KEPUSTAKAAN 5](#_Toc152698125)

[2.1 Kajian Pustaka 5](#_Toc152698126)

[2.2 Aturan Catur 6](#_Toc152698127)

[2.2.1 Aturan Dasar dan Tujuan 6](#_Toc152698128)

[2.2.2 Buah Catur 6](#_Toc152698129)

[2.2.3 Aturan Gerak Buah Catur 7](#_Toc152698130)

[2.3 Fail .PGN 8](#_Toc152698131)

[2.4 *Universal Chess Interface* (UCI) 9](#_Toc152698132)

[2.5 Notasi Gerak Catur 11](#_Toc152698133)

[2.5.1 Sistem Koordinat Papan Catur 11](#_Toc152698134)

[2.5.2 Notasi Gerak *Short Algebraic Term* 11](#_Toc152698135)

[2.5.3 Notasi Gerak *Long Algebraic Term* 13](#_Toc152698136)

[2.6 Artificial Neural Network(ANN) 13](#_Toc152698137)

[2.6.1 Recurrent Neural Network(RNN) 13](#_Toc152698138)

[2.6.2 Long Short Term Memory(LSTM) 14](#_Toc152698139)

[2.6.3 Lapisan Linear 18](#_Toc152698140)

[*2.7 Crossentropy Loss* 20](#_Toc152698141)

[2.8 Stockfish 20](#_Toc152698142)

[*2.9 Centipawn* dan *Centipawn Score* 20](#_Toc152698143)

[2.10 Akurasi Gerakan catur 21](#_Toc152698144)

[BAB 3 METODOLOGI 23](#_Toc152698145)

[3.1 Tipe dan Alur Penelitian 23](#_Toc152698146)

[3.2 Lokasi Penelitian 23](#_Toc152698147)

[3.3 Pengumpulan Data 23](#_Toc152698148)

[3.4 Pembersihan dan Pra-pemrosesan Data 24](#_Toc152698149)

[3.5 Pembangunan dan Pelatihan Model 24](#_Toc152698150)

[3.6 Evaluasi dan Optimasi Model 24](#_Toc152698151)

[3.7 Pembangunan Mesin Catur 24](#_Toc152698152)

[3.8 Pengujian dan Analisis Hasil 25](#_Toc152698153)

[3.9 Peralatan Pendukung 25](#_Toc152698154)

[3.9.1 Perangkat Keras 25](#_Toc152698155)

[3.9.2 Perangkat Lunak 25](#_Toc152698156)

[BAB 4 PERANCANGAN 26](#_Toc152698157)

[4.1 Perancangan Algoritma 26](#_Toc152698158)

[4.1.1 Pembersihan Data 26](#_Toc152698159)

[4.1.2 Pra-pemrosesan Data 27](#_Toc152698160)

[4.1.3 Kelas Model 29](#_Toc152698161)

[4.1.4 Pelatihan Model 30](#_Toc152698162)

[4.1.5 Mesin Catur 32](#_Toc152698163)

[4.2 Perhitungan Manual 33](#_Toc152698164)

[4.2.1 Pra-pemrosesan 34](#_Toc152698165)

[4.2.2 Inisialisasi Bobot 34](#_Toc152698166)

[4.2.3 Pelatihan Model 35](#_Toc152698167)

[4.2.4 Prediksi Data Uji 41](#_Toc152698168)

[4.3 Rancangan Pengujian 43](#_Toc152698169)

[4.3.1 *Hyperparameter Tuning* 43](#_Toc152698170)

[4.3.2 Pengujian Melawan Mesin Catur Lain 44](#_Toc152698171)

[4.3.3 Observasi Karakteristik 44](#_Toc152698172)

[BAB 5 IMPLEMENTASI 46](#_Toc152698173)

[5.1 Implementasi Algoritma 46](#_Toc152698174)

[5.1.1 Pembersihan Data 46](#_Toc152698175)

[5.1.2 Implementasi Pra-pemrosesan Data 47](#_Toc152698176)

[5.1.3 Kelas Model 49](#_Toc152698177)

[5.1.4 Pelatihan dan Pengujian Model 50](#_Toc152698178)

[5.1.5 Mesin Catur 52](#_Toc152698179)

[DAFTAR REFERENSI 55](#_Toc152698180)

DAFTAR TABEL

[Tabel 2.1 Contoh Komunikasi menggunakan Protokol UCI 9](#_Toc152698181)

[Tabel 2.2 Nama Buah Catur dan Inisialnya 12](#_Toc152698182)

[Tabel 4.1 Pembersihan Gerakan 27](#_Toc152698183)

[Tabel 4.2 Pemisahan Fitur dan Label 28](#_Toc152698184)

[Tabel 4.3 Pasca-pemrosesan Gerakan 31](#_Toc152698185)

[Tabel 4.4 Arsitektur Model untuk Perhitungan Manual 33](#_Toc152698186)

[Tabel 4.5 Pra-pemrosesan untuk Perhitungan Manual 34](#_Toc152698187)

[Tabel 4.6 Inisialisasi Bobot *Gate* LSTM 35](#_Toc152698188)

[Tabel 4.7 Inisialisasi *Hidden State* dan *Cell State* Awal 35](#_Toc152698189)

[Tabel 4.8 Inisialisasi Bobot Awal Lapisan Linear 35](#_Toc152698190)

[Tabel 4.9 Bobot Baru Lapisan Linear 38](#_Toc152698191)

[Tabel 4.10 Bobot Baru *Gate* Setelah Propagasi Mundur 41](#_Toc152698192)

[Tabel 4.11 Pasca-pemrosesan Perhitungan Manual 43](#_Toc152698193)

[Tabel 4.11 Perbandingan Nilai Sesungguhnya dan Hasil Prediksi 43](#_Toc152698194)

[Tabel 4.12 Rancangan *Hyperparameter Tuning* 44](#_Toc152698195)

[Tabel 4.13 Rancangan Pengujian Melawan Mesin Catur Lain 44](#_Toc152698196)

DAFTAR GAMBAR

[Gambar 2.1 Buah Catur 6](#_Toc149632895)

[Gambar 2.2 Buah Catur dalam Gim 6](#_Toc149632896)

[Gambar 2.3 *Castle* Pendek oleh Putih 8](#_Toc149632897)

[Gambar 2.4 *Castle* Panjang oleh Hitam 8](#_Toc149632898)

[Gambar 2.5 Contoh Isi dari FailPGN 9](#_Toc149632899)

[Gambar 2.6 e2e4 sebagai Gerakan Pertama dari Pengguna 10](#_Toc149632900)

[Gambar 2.7 g8f6 sebagai Gerakan Balasan dari Mesin 10](#_Toc149632901)

[Gambar 2.8 Sistem Koordinat Papan Catur 11](#_Toc149632902)

[Gambar 2.9 Gerakan Qh5 11](#_Toc149632903)

[Gambar 2.10 Contoh Kondisi Ambiguitas Notasi Gerak 12](#_Toc149632904)

[Gambar 2.11 Gerakan h7h5 13](#_Toc149632905)

[Gambar 2.12 Arsitektur *Recurrent Cell* 14](#_Toc149632906)

[Gambar 2.13 Arsitektur LSTM 15](#_Toc149632907)

[Gambar 3.1 Diagram Alir Penelitian 23](#_Toc149632908)

[Gambar 4.1 Arsitektur Model untuk Perhitungan Manual 33](#_Toc149632909)

# PENDAHULUAN

## Latar Belakang

Catur adalah permainan papan yang pertama kali muncul di Benua Eropa pada akhir abad ke-15. Permainan ini dimainkan oleh 2 orang oposisi dengan menggerakkan buah catur masing-masing di sebuah papan petak dengan dimensi 8x8. Tujuan utama dari permainan ini adalah untuk menyerang buah raja lawan sehingga lawan tidak memiliki gerakan legal untuk dimainkan, posisi ini disebut juga dengan *checkmate* atau sekakmat. Dalam situasi dengan kedua belah pihak tidak memungkinkan untuk melakukan sekakmat, maka permainan dinyatakan seri. (The International Chess Federation [FIDE] Handbook, 2022).

Upaya untuk membuat sebuah program komputer yang dapat bermain catur telah dimulai sejak tahun 1950-an, program ini sering juga disebut dengan *chess engine* atau *chess bot* yang berarti mesin catur. Dengan 64 petak yang terdapat dalam sebuah papan catur dan 16 buah catur dari masing-masing pihak yang terdiri dari buah raja, menteri, benteng, gajah, kuda, dan bidak, maka diperkirakan ada sebanyak kemungkinan posisi yang dapat terjadi dalam permainan catur (Shannon, C.E., 1950). Angka tersebut jauh melebihi kapasitas dan kemampuan komputasi komputer yang ada bahkan hingga saat ini, maka dari itu, membangun mesin catur dengan mempertimbangkan seluruh kemungkinan posisi yang dapat terjadi hampir mustahil untuk dilakukan.

Mengingat jumlah kemungkinan yang sangat tinggi, mesin catur pada umumnya dibangun dengan menggunakan skor untuk mengevaluasi posisi dan menentukan pihak mana yang diunggulkan dalam posisi tersebut, skor ini sering disebut sebagai *evaluation score* atau *eval score*. Pemberian skor pada mesin catur tradisional dilakukan dengan metode Alpha-Beta Search dan fungsi evaluasi yang dibuat secara eksplisit oleh pengembang mesin catur, seperti jumlah material (buah catur) yang dimiliki pemain, keamanan posisi dari buah raja, ruang gerak dari buah catur, dan lain-lain. Meskipun demikian, belakangan ini penggunaan *deep learning* dalam pengembangan mesin catur telah membuahkan hasil yang menjanjikan. Pada tahun 2017, mesin catur AlphaGo milik Google Deepmind yang mengimplementasikan *deep neural network* telah berhasil mengalahkan mesin catur terkuat pada saat itu, Stockfish, yang merupakan mesin catur tradisional. Pengembangan-pengembangan terbaru seperti Leela Chess Zero, proyek mesin catur *open source* yang dipimpin oleh Gary Linscott, dan Stockfish NNUE (*Neural Network Updatable Efficiently*) milik Hisayori Noda juga menunjukkan bahwa penggunaan *neural network* akan terus mendominasi dalam bidang mesin catur untuk beberapa waktu ke depan (Klean, D. 2022).

LSTM atau Long Short Term Memory adalah salah satu jenis n*eural network* yang dapat digunakan pada data berurutan atau *sequence data* (Selle, N. 2022), karena kemampuannya dalam mengingat informasi yang penting dan melupakan informasi yang tidak penting dalam sebuah rangkaian informasi yang berurutan (Santosa, R.D.W. 2021). Kemampuan LSTM ini berpotensi untuk dapat digunakan dalam pengembangan mesin catur, dengan menerima rangkaian gerakan buah catur yang telah dilakukan pemain sebagai masukan, kemudian melakukan prediksi gerakan berikutnya sebagai keluaran.

## Rumusan Masalah

Berdasarkan uraian di atas, maka rumusan masalah dari penelitian ini adalah sebagai berikut:

1. Bagaimana konfigurasi *hyperparameter* dari mesin catur yang dikembangkan menggunakan Long Short Term Memory (LSTM)?
2. Bagaimana kinerja mesin catur yang dikembangkan menggunakan Long Short Memory Term (LSTM)?

## Tujuan

Berdasarkan rumusan masalah sebagaimana diuraikan di atas, maka tujuan dari penelitian ini adalah sebagai berikut:

1. Mengatur konfigurasi *hyperparameter* yang memaksimalkan kinerja dari model Long Short Memory Term (LSTM) untuk digunakan dalam pengembangan mesin catur.
2. Mengukur kinerja mesin catur yang dikembangkan menggunakan Long Short Term Memory (LSTM).

## Manfaat

Penggunaan *neural network* sedang menjadi topik hangat di dunia pemrograman catur. *Neural network* sejauh ini telah membawa hasil yang menjanjikan dan potensi untuk terus mendominasi di ranah terkait. Penelitian ini berkontribusi untuk memberikan pendekatan, potensi, dan inovasi baru dalam pemrograman mesin catur berbasis *neural network,* dengan menggunakan rangkaian gerakan sebagai masukan model, dan model LSTM sebagai *move generator.*

## Batasan Masalah

Mesin catur merupakan sebuah program komputer yang penggunaannya terbatas pada *Command Line Interface* atau CLI. Tanpa adanya antarmuka untuk visualisasi papan catur, pengguna akan sulit untuk melakukan pertandingan hanya dengan mengandalkan notasi gerakan yang tidak mudah untuk dimengerti dan tidak efisien. Untuk mendapatkan visualisasi dari papan catur, mesin catur dapat digunakan dengan bantuan program antarmuka catur yang telah ada dan tersedia untuk digunakan saat ini, seperti Arena Chess, Cute Chess, atau Lichess.

## Sistematika Pembahasan

**BAB I PENDAHULUAN**

Bab ini memberikan gambaran umum dari penelitian, meliputi penguraian latar belakang masalah, rumusan masalah, tujuan dan manfaat penelitian, batasan masalah, serta sistematika pembahasan. Latar belakang masalah berisi uraian permasalahan yang menjadi alasan dilakukannya penelitian ini, beserta dengan solusi yang ditawarkan oleh penulis. Rumusan masalah berisi pernyataan masalah-masalah yang akan dijawab pada penelitian ini. Tujuan penelitian berisi tujuan yang ingin dicapai pada akhir penelitian. Manfaat penelitian berisi dampak positif dan kontribusi yang diberikan oleh penelitian ini. Batasan masalah berisi perumusan ruang lingkup penelitian. Sistematika pembahasan berisi struktur penulisan dari laporan penelitian ini.

**BAB II LANDASAN KEPUSTAKAAN**

Bab ini berisi uraian mengenai dasar teori yang digunakan pada penelitian ini, termasuk penelitian serupa yang pernah dilakukan sebelumnya. Penelitian serupa akan dibahas pada subbab kajian pustaka, dan teori-teori yang digunakan akan dibahas pada subbabnya masing-masing yang meliputi aturan catur, *Universal Chess Interface, Portable Game Notation,* notasi gerak catur, dan *Artificial Neural Network.*

**BAB III METODOLOGI**

Bab ini berisi uraian mengenai kumpulan metode, cara, dan tahapan penelitian yang akan dilakukan, yang tersusun secara berurutan. Isi dari bab ini meliputi pengumpulan data, pembersihan dan pra-pemrosesan data, pembangunan dan pelatihan model, evaluasi dan optimasi model, pembangunan mesin catur, serta pengujian dan analisis hasil.

**BAB IV PERANCANGAN**

Bab ini berisi penjelasan mengenai rancangan penelitian, sesuai dengan metodologi penelitian yang dijelaskan pada bab sebelummnya. Perancangan yang dilakukan mencakup hal-hal seperti *pseudocode* dari kode program yang dibutuhkan, perhitungan manual dari metode metode yang digunakan, dan sistematika pengujian yang akan dilakukan. Rancangan yang dirumuskan pada bab ini akan menjadi landasan untuk eksekusi dan implementasi penelitian yang dilakukan pada bab berikutnya

**BAB V IMPLEMENTASI**

Bab ini berisi dokumentasi dan penjelasan dari proses pelaksanaan metodologi penelitian yang telah direncanakan pada bab sebelumnya. Isi dari bab ini melibatkan implementasi kode program untuk pembersihan dan pra-pemrosesan data, pembangunan model LSTM yang dimulai dari pemilihan arsitektur hingga evaluasi dan optimasi model, pembangunan mesin catur dengan protokol UCI, serta pengujian mesin catur.

**BAB VI HASIL DAN PEMBAHASAN**

Bab ini berisi penjelasan mengenai hasil yang didapatkan dari bab sebelumnya, beserta dengan analisis, pembahasan, dan evaluasinya. Analisis dan pembahasan dilakukan sebagai upaya untuk memahami sebab dan alasan dari hasil yang didapatkan. Evaluasi dilakukan untuk meninjau baik atau buruknya hasil yang didapatkan, beserta dengan solusi perbaikannya.

**BAB VII PENUTUP**

Bab ini berisi kesimpulan dari analisis, pembahasan, dan evaluasi hasil yang telah dilakukan pada bab sebelumnya, serta saran penulis untuk penelitian-penelitian terkait berikutnya. Kalimat kesimpulan akan ditulis sebagai jawaban dari rumusan masalah yang telah dirumuskan sebelumnya.

# LANDASAN KEPUSTAKAAN

## Kajian Pustaka

Panchal, H., Mishra, S. dan Shrivastava, V. (2020) melakukan penelitian mengenai prediksi gerakan catur menggunakan Convolutional Neural Network atau CNN. Penelitian ini menggunakan sebanyak 1.500.000 posisi papan catur sebagai data penelitian, masing-masing dari posisi tersebut direpresentasikan sebagai matriks berukuran 8x8x14. 8x8 merepresentasikan 64 petak yang ada pada papan catur, dan 14 merupakan representasi dari 6 jenis buah catur milik putih, 6 jenis buah catur milik hitam, dan 2 sisanya merepresentasikan petak yang dapat diserang oleh putih dan hitam. Dalam penelitian ini, digunakan 2 buah *network,* *network* pertama bertanggung jawab untuk menerima matriks representasi posisi berukuran 8x8x14 sebagai masukan, kemudian menentukan petak *from\_square*, yaitu petak di mana buah catur yang akan digerakkan berdiri. *Network* kedua juga menerima matriks representasi posisi papan yang sama, namun bertugas untuk menentukan petak *to\_square,* yaitu petak tujuan dari buah catur yang akan digerakkan.

Pengujian sistem pada penelitian ini dilakukan dengan menandingkan sistem dengan mesin catur bernama Stockfish, sebuah mesin catur tradisional yang mengimplementasikan algoritma minimax dan Alpha-Beta Pruning, serta dianggap sebagai salah satu mesin catur terkuat. Pengujian dilakukan sebanyak 100 pertandingan, dan hasil yang didapatkan adalah 3% dari seluruh pertandingan dimenangkan oleh sistem, 2% dari seluruh pertandingan berakhir dengan hasil seri, dan 95% dari seluruh pertandingan dimenangkan oleh Stockfish. Karakteristik permainan sistem yang ditemukan adalah preferensi sistem terhadap ruang gerak buah catur, bahkan, sistem rela mengorbankan beberapa buah catur untuk ditangkap oleh lawan demi membuka ruang gerak buah catur lainnya. Sistem juga mampu mengidentifikasi taktik-taktik sederhana seperti *fork* (menyerang 2 buah lawan pada saat yang bersamaan) dan *pin* (lihat poin 9.2 pada subbab 2.2.3). Meski begitu, sistem gagal dalam menyadari taktik lawan dan melindungi petak krusial, sehingga berujung pada kekalahan.

Kesimpulan dari penelitian ini adalah prediksi gerakan catur menggunakan model CNN mampu mengidentifikasi berbagai taktik dasar seperti identifikasi buah catur untuk ditangkap, melakukan pengembangan posisi buah catur, dan merancang skenario untuk mencapai hasil seri. Kekurangan dari model ini adalah kurangnya kemampuan model dalam melakukan pergerakan yang inovatif dan kreatif untuk memperbesar peluang kemenangan.

## Aturan Catur

Mengutip dari International Chess Federation (FIDE), aturan dari permainan catur dapat dilihat pada bagian selanjutnya.

### Aturan Dasar dan Tujuan

1. Permainan catur dimainkan oleh dua oposisi dengan menggerakkan buah catur masing-masing di sebuah papan yang dinamakan papan catur.
2. Pihak dengan buah catur yang berwarna terang, biasa juga disebut dengan putih, melakukan gerakan pertama, kemudian kedua pihak melakukan gerakan secara bergantian.
3. Tujuan permainan adalah untuk menyerang buah raja lawan sehingga lawan tidak memiliki gerakan legal untuk dimainkan. Posisi ini dinamakan dengan *checkmate* atau sekakmat. Pihak yang berhasil melakukan sekakmat dinyatakan memenangkan pertandingan.
4. Dalam kondisi dengan kedua pihak tidak memungkinkan untuk melakukan sekakmat, maka pertandingan dinyatakan seri.

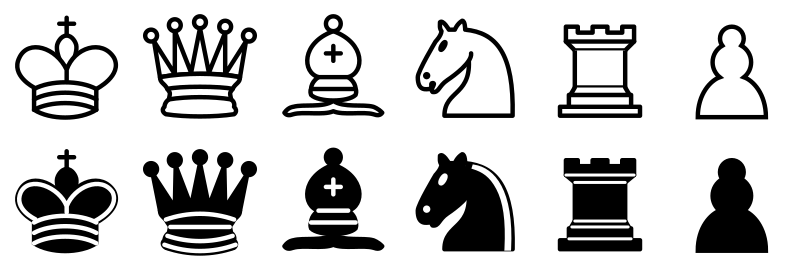
### Buah Catur

Masing-masing pihak memiliki 16 buah catur untuk dimainkan yang terdiri dari 8 buah bidak/pion, 1 buah raja, 1 buah menteri, 2 buah benteng, 2 buah gajah, dan 2 buah kuda. Buah catur dari kedua pihak dibedakan berdasarkan warna menjadi warna terang dan warna gelap. Bentuk dari masing-masing buah catur dapat dilihat pada Gambar 2.1. dan Gambar 2.2.



Gambar 2.1 Buah Catur

Sumber: Frank A. (2004))



Gambar 2.2 Buah Catur dalam Gim

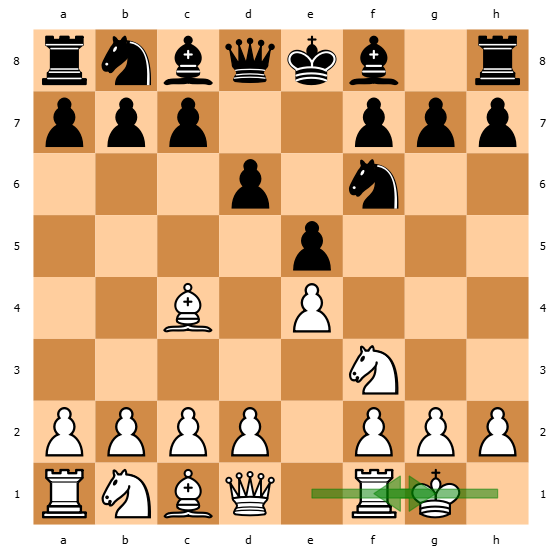
Sumber: jurgenwesterhof (2014))

Keterangan, dari kiri ke kanan: Raja, Menteri, Gajah, Kuda, Benteng, Pion.

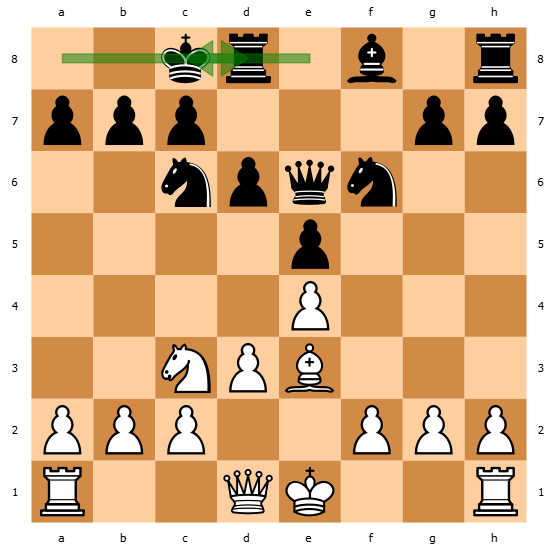
### Aturan Gerak Buah Catur

Mengutip dari FIDE, terdapat berbagai aturan mengenai pergerakan buah catur, dengan perincian sebagai berikut:

1. Dilarang menggerakkan buah catur ke petak yang ditempati oleh buah lain dengan warna yang sama.
   1. Jika buah digerakkan ke petak yang ditempati oleh buah catur lawan, maka buah catur lawan dinyatakan tertangkap dan dikeluarkan dari papan permainan.
   2. Buah catur dinyatakan “menyerang” buah lawan jika buah tersebut dapat menangkap buah lawan pada langkah berikutnya.
   3. Buah catur tetap dinyatakan menyerang buah lawan, meskipun buah tersebut tidak dapat bergerak karena sedang melindungi raja dari serangan.
2. Gajah dapat bergerak ke petak mana pun sepanjang diagonal dari petak di mana dia berdiri.
3. Benteng dapat bergerak ke petak mana saja secara vertikal (*file)* atau horizontal (*rank)* dari petak di mana dia berdiri.
4. Menteri dapat bergerak ke petak mana saja sepanjang diagonal, *file,* atau *rank* dari petak di mana dia berdiri.
5. Saat bergerak, gajah, benteng, maupun menteri tidak dapat melewati buah lain yang menghalangi.
6. Kuda dapat bergerak sebanyak 2 petak secara horizontal dan 1 petak secara vertikal, atau 2 petak secara vertikal dan 1 petak secara horizontal, menyerupai bentuk huruf “L”.
7. Bidak/pion:
   1. Pion dapat bergerak maju ke petak tepat di depannya jika petak tersebut tidak ditempati, atau;
   2. Pada gerakan pertamanya, pion dapat bergerak maju sebagaimana poin 7.2, atau bergerak maju sebanyak 2 petak, jika kedua petak tersebut tidak ditempati.
   3. Pion dapat menangkap buah catur lawan yang berdiri pada petak diagonal di depannya.
   4. Saat pion digerakkan ke *rank* terjauh dari posisi awal, pion tersebut harus diganti dengan salah satu dari menteri, kuda, gajah, atau benteng di petak terakhir pion berdiri. Gerakan ini disebut dengan promosi pion.
8. Terdapat 2 cara raja dapat bergerak:
9. Bergerak ke petak mana pun yang berdampingan dengan petak di mana dia berdiri, atau;
10. Dengan “*castling*”, yaitu gerakan saat raja bergerak sebanyak 2 petak ke arah benteng, dan benteng berpindah ke petak yang baru saja ditinggalkan oleh raja. *Castle* hanya dapat dilakukan jika tidak ada buah yang berdiri di antara raja dan benteng, dan tidak ada buah lawan yang menyerang petak-petak diantara raja dan benteng. *Castle* yang dilakukan ke sisi raja disebut dengan *short castle* atau *castle* pendek*,* sedangkan *castle* yang dilakukan ke sisi menteri sering disebut juga dengan *long castle* atau *castle* panjang*.*



Gambar 2.3 *Castle* Pendek oleh Putih



Gambar 2.4 *Castle* Panjang oleh Hitam

1. Raja dalam serangan/sekak:
2. Raja dinyatakan dalam sekak jika diserang oleh 1 atau lebih buah lawan.
3. Buah catur tidak boleh digerakkan jika gerakan tersebut membuka serangan terhadap raja dengan warna yang sama, atau jika gerakan tersebut membiarkan raja dalam kondisi sekak. Buah dalam kondisi pertama sering disebut juga dengan “*pinned piece”* atau “buah terjepit”.

## Fail .PGN

*Portable Game Notation* atau PGN adalah format fail yang diperkenalkan oleh Steven J. Edwards pada tahun 1994, dan dipopulerkan melalui grup berita rec.chess.com. PGN didesain sebagai representasi dari satu atau lebih pertandingan catur menggunakan teks ASCII. PGN menyimpan berbagai informasi mengenai sebuah pertandingan catur, contoh data pertandingan yang disimpan dalam format PGN dapat dilihat pada Gambar 2.5.

A computer screen shot of a black screen

Description automatically generated

Gambar 2.5 Contoh Isi dari FailPGN

Sumber: database.lichess.org

Dua baris terakhir dalam Gambar 2.5 merupakan rangkaian gerakan yang dilakukan di pertandingan terkait. Rangkaian gerakan tersebut dicatat dalam notasi *Short Algebraic Term,* penjelasan lebih lanjut mengenai notasi gerakan catur akan dijelaskan pada Subbab 2.5.

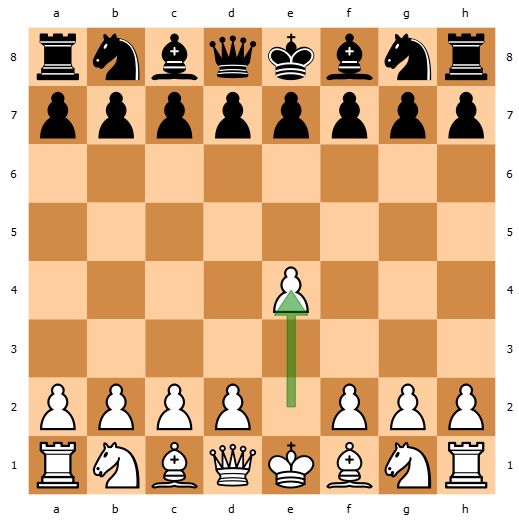
## *Universal Chess Interface* (UCI)

*Universal Chess Interface* atau UCI adalah protokol komunikasi yang memungkinkan sebuah mesin catur untuk dapat berkomunikasi dengan sebuah antarmuka catur. Protokol ini dirilis pada bulan April tahun 2004 oleh Stefan Meyer-Kahlen. Komunikasi antara mesin catur dan antarmuka catur dilakukan secara dua arah melalui *stdin* dan *stdout.* Pada Tabel 2.1 dapat dilihat contoh komunikasi yang dilakukan antara antarmuka catur dengan mesin catur saat memulai sebuah pertandingan, kemudian melakukan 1 gerakan untuk masing-masing hitam dan putih.

Tabel 2.1 Contoh Komunikasi menggunakan Protokol UCI

|  |  |  |
| --- | --- | --- |
| **No.** | **Masukan dari Antarmuka** | **Respon yang Diharapkan** |
| 1. | uci | Uciok |
| 2. | isready | Readyok |
| 3. | ucinewgame |  |
| 4. | Position startpos moves e2e4 |  |
| 5. | isready | Readyok |
| 6. | go infinite | bestmove g8f6 |

Komunikasi dimulai dengan masukan “uci” dari antarmuka ke mesin yang merupakan perintah kepada mesin catur untuk masuk ke mode UCI. Mesin membalas dengan keluaran “uciok” yang menyatakan bahwa mesin telah berhasil masuk ke mode UCI. Antarmuka kemudian memberikan masukan “isready” untuk memastikan bahwa mesin telah siap untuk memulai pertandingan. Mesin membalas dengan keluaran “readyok”, menyatakan bahwa mesin telah siap. Pertandingan dimulai dengan masukan “ucinewgame”, dan gerakan pertama dilakukan oleh pengguna melalui antarmuka. Antarmuka kemudian menginformasikan gerakan yang telah dilakukan oleh pengguna kepada mesin dengan masukan “position startpos moves e2e4”.



Gambar 2.6 e2e4 sebagai Gerakan Pertama dari Pengguna

Antarmuka kemudian memastikan kembali bahwa mesin berada pada kondisi siap dengan masukan “isready”, yang dibalas oleh mesin melalui keluaran “readyok”, menyatakan bahwa mesin siap untuk melakukan gerakan. Antarmuka mempersilakan mesin untuk melakukan gerakan dengan masukan “go infinite”. Kata “infinite” menyatakan bahwa tidak ada batasan waktu untuk mesin melakukan gerakan, kata ini dapat diganti menjadi batas waktu untuk melakukan gerakan yang telah ditentukan sebelumnya. Mesin kemudian melakukan pertimbangan gerakan apa yang terbaik untuk dilakukan. Setelah memutuskan gerakan apa yang akan dilakukan, mesin menginformasikannya kepada antarmuka dengan keluaran “bestmove g8f6”.

A chess board with chess pieces

Description automatically generated

Gambar 2.7 g8f6 sebagai Gerakan Balasan dari Mesin

Dalam uraian sebelumnya, “e2e4” dan “g8f6” merupakan gerakan yang dituliskan dalam notasi *Long Algebraic Term.* Penjelasan lebih lanjut mengenai notasi gerakan catur akan dibahas pada Subbab berikutnya.

## Notasi Gerak Catur

### Sistem Koordinat Papan Catur

Terdapat 64 petak pada sebuah papan catur. Masing-masing petak diidentifikasi dengan koordinat sebagaimana dapat dilhat pada Gambar 2.8.

A grid of squares with letters and numbers

Description automatically generated

Gambar 2.8 Sistem Koordinat Papan Catur

Sumber: FIDE (2022)

Baris dalam papan catur disebut juga dengan *file,* sebagai contoh, barisan petak dari b1 sampai dengan b8 disebut dengan “*b file”.* Kolom dalam papan catur disebut juga sebagai *rank,* sebagai contoh, kolom a7 sampai dengan h7 disebut dengan “*7th rank”* atau *rank* ketujuh.

### Notasi Gerak *Short Algebraic Term*

Notasi *Short Algebraic Term* adalah notasi yang digunakan pada format failPGN. Notasi ini menggunakan inisial buah catur dalam Bahasa Inggris dan koordinat petak tujuan untuk menyatakan gerakan. Sebagai contoh, “Qh5” dapat dibaca pula sebagai “*Queen to h5”* atau “menteri ke h5”, sebagaimana dapat dilihat pada Gambar 2.9.

A chess board with chess pieces

Description automatically generated

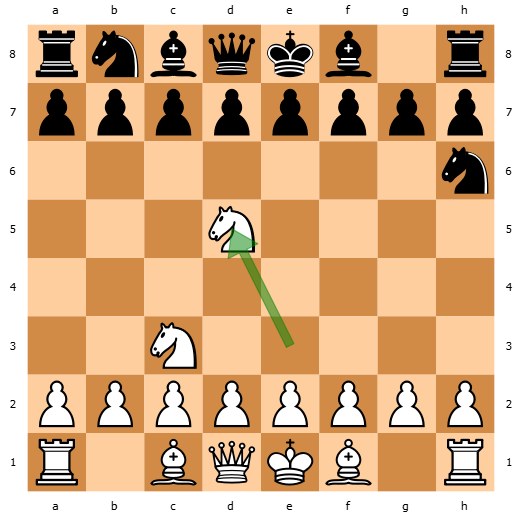
Gambar 2.9 Gerakan Qh5

Perincian nama buah catur dalam Bahasa Indonesia, Bahasa Inggris, dan inisial yang digunakan untuk merepresentasikannya dalam notasi *Short Algebraic Term* dapat dilihat pada Tabel 2.2.

Tabel 2.2 Nama Buah Catur dan Inisialnya

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Nama Buah dalam Bahasa Indonesia** | **Nama Buah dalam Bahasa Inggris** | **Inisial yang digunakan** |
| 1. | Raja | *King* | K |
| 2. | Menteri | *Queen* | Q |
| 3. | Benteng | *Rook* | R |
| 4. | Gajah | *Bishop* | B |
| 5. | Kuda | *Knight* | N |
| 6. | Pion | *Pawn* | - |

Pada tabel tersebut pion tidak memiliki inisial, hal tersebut dikarenakan pada notasi *Short Algebraic Term* gerakan “pion ke c3” hanya akan ditulis sebagai “c3”. Terdapat beberapa notasi lain yang digunakan untuk kondisi tertentu. Simbol “+” digunakan jika gerakan terkait menghasilkan sekak untuk raja lawan (contoh: Ra8+), huruf “x” digunakan jika gerakan terkait menangkap buah catur lawan (contoh: Nxc7), dan simbol “#” digunakan jika gerakan terkait menghasilkan sekakmat (contoh: Qb2#). Pada kondisi dengan 2 buah catur yang identik dapat bergerak ke petak yang sama, maka digunakan koordinat dari *file* atau *rank* posisi awal untuk mengidentifikasi buah mana yang digerakkan.



Gambar 2.10 Contoh Kondisi Ambiguitas Notasi Gerak

Pada Gambar 2.10, terdapat 2 buah kuda yang dapat bergerak ke d5, yaitu kuda di c3 dan e3, maka gerakan pada gambar ditulis sebagai “Ne3d5”, mengindikasikan bahwa gerakan dilakukan oleh kuda yang sebelumnya berada pada petake3, sebaliknya, jika kuda pada petakc3 yang melakukan gerakan, maka gerakan akan ditulis sebagai “Nc3e5” (FIDE, 2022).

### Notasi Gerak *Long Algebraic Term*

Notasi *Long Algebraic Term* tidak menggunakan inisial buah catur untuk mengidentifikasi buah mana yang digerakkan, melainkan menggunakan koordinat petak sebelum buah digerakkan. Sebagai contoh, gerakan “h7h5” menyatakan bahwa buah catur yang sebelumnya berada di h7, kini digerakkan ke h5. Notasi ini adalah notasi yang digunakan dalam protokol UCI, sehingga diperlukan penyesuaian notasi untuk memenuhi kebutuhan data penelitian ini.

A chess board with chess pieces

Description automatically generated

Gambar 2.11 Gerakan h7h5

## Artificial Neural Network(ANN)

Artificial Neural Networkatau ANN, sering disebut juga sebagai Jaringan Syaraf Tiruan atau JST, adalah sebuah struktur yang tersusun dari unit-unit pemroses informasi sederhana yang saling terkoneksi, yang mampu melakukan komputasi paralel yang sangat besar untuk mengolah data dan merepresentasikan pengetahuan. (Hecht-Nielsen, 1990; Schalkoff 1997). Meskipun pembangunan ANN didasarkan pada cara kerja neuron biologis, namun tujuan dari ANN bukanlah untuk mereplika neuron biologis, melainkan untuk memanfaatkan apa yang telah diketahui mengenai cara kerja neuron biologis, kemudian menerapkannya untuk menyelesaikan masalah-masalah kompleks. Keunggulan dari ANN adalah karakteristik pengolahan data yang menyerupai sistem biologis, seperti kemampuan generalisasi informasi, kemampuan untuk mengolah data secara paralel, toleransi terhadap kesalahan, dan kemampuan untuk menangani data dengan informasi yang kabur (Jain et al., 1996).

### Recurrent Neural Network(RNN)

Recurrent Neural Networkatau RNN adalah sebutan untuk model ANN yang menggunakan *recurrent cell* dalam arsitekturnya (Yu, Y., 2019). Penggunaan *recurrent cell* membuat RNN memiliki karakteristik alur informasi dua arah atau *bi-directional information flow* menggunakan mekanisme yang disebut sebagai *feedback loops* yang memungkinkan keluaran dari sebuah *node* untuk dapat disimpan atau diingat oleh model dan mempengaruhi perhitungan dari masukan berikutnya.

A diagram of a mathematical equation

Description automatically generated with medium confidence

Gambar 2.12 Arsitektur *Recurrent Cell*

Mekanisme *recurrent cell* dapat ditulis sebagai ekspresi matematika sebagaimana pada Persamaan 2.1 dan 2.2.

(2.1)

(2.2)

Keterangan:

= Informasi yang disimpan/informasi rekurens

= fungsi aktivasi sigmoid

= bobot h

= informasi rekurens sebelumnya

= bobot x

= data masukan

= *bias*

=keluaran

### Long Short Term Memory(LSTM)

Long ShortTerm Memory atau LSTM adalah salah satu bentuk RNN dengan kemampuan dan karakteristik khusus untuk mengingat informasi dalam periode waktu yang lama. LSTM diperkenalkan oleh Hochreiter & Schmidhuber pada tahun 1997 untuk menyelesaikan masalah *exploding* dan *vanisihing gradient* yang marak ditemui pada RNN biasa, yaitu kondisi dengan gradien yang membesar atau mengecil secara eksponensial seiring bertambahnya iterasi, dikarenakan perkalian bobot yang berulang kali (Hochreiter, S. dan Schmidhuber, J., 1997). LSTM memperkenalkan *cell state* atau memori jangka panjang yang tidak dipengaruhi secara langsung oleh bobot, sebuah konsep yang memungkinkan terjadinya gradien yang konstan, serta penyimpanan informasi untuk jangka waktu yang lama. Komponen dari LSTM terdiri dari *cell state* sebagai penyimpan informasi jangka panjang, *hidden state* sebagai penyimpan informasi jangka pendek, *input gate, candidate gate, forget gate,* dan *output gate*.

A diagram of a computer

Description automatically generated

Gambar 2.13 Arsitektur LSTM

#### Propagasi Maju

Propagasi maju adalah tahap pertama dari lapisan LSTM. Pada tahap ini, data masukan diproses secara berurutan melalui komponen-komponen lapisan. *Forget gate* adalah komponen pertama model LSTM. *Forget gate* bertanggung jawab untuk memutuskan seberapa banyak dari informasi pada *cell state* sebelumnya yang akan diingat untuk *step* berikutnya*.* Perhitungan *forget gate* dilakukan menggunakan fungsi aktivasi *sigmoid* sebagaimana dapat dilihat pada Persamaan 2.3.

Persamaan *forget gate*

(2.3)

keterangan*:*

=nilai *forget gate*

*σ* = fungsi aktivasi *sigmoid*

=bobot h pada *forget gate*

=bobot x pada *forget gate*

= *input* ke-t

= *hidden state/*memori jangka panjang sebelumnya

= *bias* dari *forget gate*

Perhitungan dari *forget gate* akan menghasilkan angka pada rentang 0 sampai dengan 1, dengan nilai 0 merepresentasikan bahwa informasi *cell state* sebelumnyaakan dilupakan sepenuhnya, dan nilai 1 merepresentasikan bahwa informasi *cell state* sebelumnya akan diingat seluruhnya. Tahap berikutnya adalah perhitungan *candidate gate* yang bertanggung jawab untuk membuat informasi baru untuk ditambahkan dan ke *cell state*, informasi baru ini disebut juga sebagai *candidate memory* atau memori kandidat. Perhitungan pada *candidate gate* ini dilakukan menggunakan fungsi aktivasi tanh sebagaimana dapat dilihat pada Persamaan 2.4.

Persamaan *candidate gate*

(2.4)

Keterangan:

= nilai *candidate memory/candidate gate*

= bobot h pada *candidate gate*

= bobot x pada *candidate gate*

= bias pada *candidate gate*

Tahap berikutnya adalah *input gate* yang bertanggung jawab untuk memutuskan berapa persen dari *candidate memory* yang akan ditambahkan ke *cell state*. Perhitungan pada *input gate* dilakukan menggunakan fungsi aktivasi sigmoid, sebagaimana dapat dilihat pada Persamaan 2.5.

Persamaan *input gate*

(2.5)

Keterangan:

= nilai *input gate*

= bobot h pada *input gate*

= bobot x pada *input gate*

= bias pada *input gate*

Sama seperti pada *forget gate*, hasil perhitungan dari *input gate* akan menghasilkan nilai dalam rentang 0 sampai dengan 1, dengan nilai 0 merepresentasikan bahwa *candidate memory* tidak akan ditambahkan ke *cell state* sama sekali, dan nilai 1 merepresentasikan bahwa seluruh informasi *candidate memory* akan ditambahkan ke *cell state*. Pembaruan nilai *cell state* kemudian dilakukan menggunakan Persamaan 2.6.

Persamaan *cell state* baru

(2.6)

Keterangan:

= nilai *cell state* baru

= nilai *cell state* sebelumnya

Tahap selanjutnya adalah *output gate* yang nantinya akan dikalikan dengan *cell state* untuk memperbarui nilai *hidden state*. Perhitungan pada tahap ini menggunakan fungsi aktivasi aktivasi sigmoid, sebagaimana dapat dilihat pada Persamaan 2.7.

Persamaan *output gate*

(2.7)

Keterangan:

= nilai *output gate*

Tahap terakhir dari model LSTM adalah memperbarui *hidden state*, yang sekaligus berperan sebagai keluaran dari lapisan LSTM. Perhitungan pada tahap ini menggunakan fungsi aktivasi *tanh*, sebagaimana dapat dilihat pada Persamaan 2.8.

Persamaan *hidden state* baru

(2.8)

Keterangan:

= nilai *output gate*

= nilai *hidden state* baru dan keluaran model LSTM

#### Propagasi Mundur

Propagasi mundur atau *backpropagation* dilakukan untuk memperbarui bobot setiap gate menggunakan algoritma *Stochastic Gradient Descent* atau SGD. Tahap ini diawali dengan mengambil turunan parsial dari fungsi *loss* yang digunakan. Nilai turunan tersebut kemudian akan digunakan untuk menghitung turunan parsial *loss* terhadap *hidden state* dan *cell state,* yang dihitung menggunakan Persamaan 2.9 dan 2.10.

Persamaan turunan *loss* terhadap *hidden state* dan *cell state*

(2.9)

(2.10)

Keterangan:

= Turunan parsial *loss* terhadap *input*

= Turunan parsial *loss* terhadap *hidden state* saat t = 1

= Turunan parsial *loss* terhadap *cell state* saat t=1

Selanjutnya, dilakukan perhitungan nilai turunan parsial *loss* terhadap masing-masing *gate*, yaitu *forget gate, input gate, candidate gate,* dan *output gate.* Persamaan untuk melakukan perhitungan tersebut dapat dilihat pada Persamaan 2.11 sampai dengan Persamaan 2.14.

Persamaan turunan parsial *loss* terhadap setiap *gate*

(2.11)

(2.12)

(2.13)

(2.14)

Keterangan:

= Turunan parsial *loss* terhadap *candidate gate* saat t = 1

= Turunan parsial *loss* terhadap *input gate* saat t=1

= Turunan parsial *loss* terhadap *forget gate* saat t = 1

= Turunan parsial *loss* terhadap *output gate* saat t=1

Terakhir, perhitungan gradien dan pembaruan bobot untuk masing-masing *gate* dilakukan dengan menerapkan SGD, sebagaimana dapat dilihat pada Persamaan 2.15 dan Persamaan 2.16.

Persamaan gradien dan pembaruan bobot *gate*

(2.15)

(2.16)

Keterangan:

= Bobot baru *gate*

= Bobot lama *gate*

= *Learning rate*

=Turunan *loss* terhadap *gate* saat t=1

= Masukan model saat t=1

= Gradien *gate*

### Lapisan Linear

Lapisan linear, sering disebut juga sebagai *linear layer, dense layer,* atau *fully-connected layer* adalah jenis lapisan paling sederhana pada JST. Lapisan ini menerapkan transformasi linear dengan terhadap data yang masuk dengan mengalikannya dengan bobot pada neuron.

#### Propagasi Maju

Propagasi maju lapisan linear hanya melibatkan 1 perhitungan, yaitu mengalikan data masukan dengan bobot masing-masing neuron, kemudian ditambahkan dengan bias, sebagaimana dapat dilihat pada Persamaan 2.17.

(2.17)

Keterangan:

= Hasil tranformasi linear

= Data masukan

= Bobot masing-masing neuron

= Bias

#### Propagasi Mundur

Propagasi mundur lapisan linear melibatkan perhitungan gradien bobot dan algoritma SGD untuk memperbarui bobot dari lapisan linear, sebagaimana dapat dilihat pada Persamaan 2.17 dan 2.18.

(2.18)

(2.19)

Keterangan:

= Gradien bobot

= Turunan parsial *loss* terhadap keluaran

= Turunan parsial keluaran terhadap bobot

= Bobot baru

= Bobot lama

= *Learning rate*

Persamaan 2.18 digunakan untuk menghitung turunan parsial *loss* terhadao bobot menggunakan konsep turunan berantai, sedangkan Persamaan 2.19 digunakan memperbarui bobot menggunakan algoritma SGD. Persamaan 2.18 dapat disederhanakan menjadi Persamaan 2.18 jika menggunakan *loss* dengan tipe *Mean Squared Error* atau MSE.

(2.20)

Keterangan:

= Label

= Nilai keluaran

= Data masukan

## *Crossentropy Loss*

*Crossentropy loss* adalah salah satu fungsi *loss* yang dapat digunakan pada permasalahan klasifikasi dengan lebih dari 1 kelas. Fungsi *loss* ini menerima hasil prediksi berupa *logits* dengan panjang C yang merupakan banyak kelas, dan label berupa indeks dari kelas sesungguhnya. Perhitungan *loss* dilakukan dengan menggunakan fungsi *ln* atau logaritma natural, sebagaimana dapat dilihat pada Persamaan 2.21.

(2.21)

Keterangan:

= Logaritma natural

= Eksponen dari *logit* pada indeks ke-x

= Jumlah kelas

= Eksponen dari *logit* pada indeks ke-n

## Stockfish

Stockfish adalah mesin catur yang dianggap sebagai mesin catur terkuat saat ini. Stockfish berawal dari sebuah proyek *open source* mesin catur bernama Glaurung yang dimulai oleh Tord Romstad. Kode dari Glaurung versi 2.1 kemudian dimodifikasi oleh Marco Costabala, yang kemudian diperkenalkan sebagai Stockfish 1.0. Joona Kiiski kemudian bergabung dalam proyek pengembangan Stockfish, dan proyek pengembangan Glaurung perlahan-lahan ditinggalkan. Sementara itu, Stockfish dengan cepat berkembang menjadi salah satu mesin catur terkuat di dunia. Hingga saat ini, Stockfish telah memenangkan kejuaraan mesin catur internasional *Top Chess Engine Championship* atau TCEC sebanyak 14 kali. Bahkan sejak kompetisi tersebut diadakan kembali pada tahun 2013, Stockfish selalu merebut gelar juara pertama atau kedua, kecuali pada musim ke-10 pada tahun 2017, di mana Stockfish merebut gelar juara ketiga. Hal tersebut tidak bertahan lama karena mesin catur Houdini yang merebut gelar juara pertama, didiskualifikasi karena menggunakan bagian kode Stockfish tanpa izin, sehingga melepas gelar juara kedua untuk diberikan ke tangan Stockfish.

Kinerja Stockfish yang sangat baik ini juga membuat Stockfish dijadikan sebagai acuan dalam perhitungan akurasi gerakan pada berbagai platform catur seperti chess.com dan lichess.org. Penjelasan mengenai perhitungan akurasi gerakan catur akan dijelaskan pada Subbab 2.9.

## *Centipawn* dan *Centipawn Score*

*Centipawn* merupakan satuan yang digunakan untuk mengukur keunggulan yang dimiliki pemain. 1 *centipawn* setara dengan 1/100 buah pion, sehingga keunggulan pemain sebanyak 100 *centipawn* menyatakan bahwa keunggulan pemain tersebut setara dengan memiliki 1 buah pion lebih banyak daripada lawan. Keunggulan yang dinyatakan dalam satuan *centipawn* disebut juga sebagai *centipawn score*. Skor ini dinyatakan dalam sudut pandang putih, yang berarti nilai *centipawn score* sebesar +100 menyatakan bahwa putih unggul sebanyak 100 *centipawn,* sedangkan nilai *centipawn score* sebesar -100 menyatakan bahwa hitam unggul sebanyak 100 *centipawn* (Mehta, F. et al, 2020)*.*

*Centipawn* dan *centipawn score* menjadi dasar dari berbagai metrik untuk mengukur kinerja pemain catur, seperti *centipawn loss* yang mengukur perubahan *centipawn score* untuk setiap gerakan yang dilakukan pemain, dan *Average Centipawn Loss* atau ACPL yang menghitung rata-rata *centipawn loss* seorang pemain pada sebuah pertandingan.

## Akurasi Gerakan catur

Akurasi dalam catur menyatakan seberapa baik gerakan yang dilakukan dalam sebuah pertandingan, akurasi 0% pada akhir pertandingan menyatakan bahwa semua gerakan yang dilakukan pada pertandingan tersebut merupakan gerakan yang buruk, sedangkan akurasi 100% pada akhir pertandingan menyatakan bahwa semua gerakan yang dilakukan pada pertandingan tersebut merupakan gerakan terbaik. Terdapat banyak cara untuk menghitung akurasi gerakan, penelitian ini akan menggunakan fitur *analyze board* milik lichess.org untuk menghitung akurasi gerakan mesin catur yang dibangun. Lichess.org melakukan perhitungan tersebut menggunakan Persamaan 2.20 dan 2.21.

(2.22)

(2.23)

Keterangan:

= peluang menang pada gerakan ke-t

= Akurasi gerakan ke-t

= *centipawn score* pada gerakan ke-t

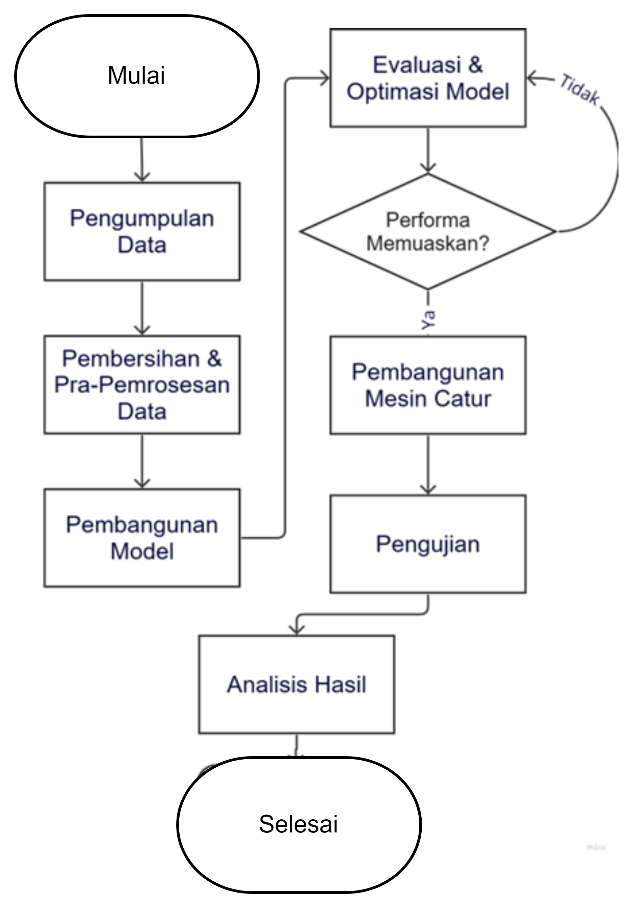
Persamaan tersebut menghitung akurasi dari sebuah gerakan berdasarkan perubahan peluang kemenangan setelah gerakan tersebut dilakukan. Gerakan yang menyebabkan penurunan peluang kemenangan yang signifikan akan diberikan nilai akurasi yang rendah, sedangkan gerakan yang tidak menyebabkan penurunan peluang kemenangan atau penurunan yang tidak signifikan, akan diberikan nilai akurasi yang tinggi. Penjelasan sebelumnya menggunakan istilah “penurunan” karena peluang kemenangan tidak dapat bertambah setelah melakukan sebuah gerakan, bertambahnya peluang kemenangan hanya dapat terjadi jika lawan melakukan kesalahan.

*centipawn score* yang digunakan dalam perhitungan tersebut merupakan *centipawn score* yang dihitung oleh Stockfish dengan mengevaluasi posisi pada gerakan ke-t untuk menentukan pihak mana yang memiliki keunggulan. Penggunaan Stockfish untuk menghitung *centipawn score* membuatgerakan yang dianggap sebagai “gerakan terbaik” pada dasarnya adalah opini Stockfish yang berkata “dalam posisi ini, aku akan melakukan gerakan ini”.

# METODOLOGI

## Tipe dan Alur Penelitian

Penelitian ini melakukan pembangunan mesin catur dengan menggunakan model LSTM sebagai *move generator*, dengan tujuan untuk menganalisis keefektifan model dalam melakukan gerakan yang bermakna, untuk memperbesar peluang kemenangan. Jenis dan tipe dari penelitian ini adalah nonimplementatif-analisis, dengan alir penelitian sebagaimana pada Gambar 3.1.



Gambar 3.1 Diagram Alir Penelitian

## Lokasi Penelitian

Penelitian pengembangan mesin catur menggunakan Long short Term Memory (LSTM) berlokasi di lingkungan laboratorium riset Fakultas Ilmu Komputer Universitas Brawijaya.

## Pengumpulan Data

Data yang digunakan pada penelitian ini bersifat data sekunder yang diambil dari halaman web database.lichess.org, sebuah *open database* yang menyediakan data berupa perincian dari seluruh pertandingan catur yang dilakukan di platform tersebut sejak bulan Mei tahun 2013 sampai dengan sekarang. Data disediakan dalam format .PGN atau *Portable Game Notifier,* dan rangkaian gerakan pertandingan tercatat dalam notasi *Short Algebratic Notation.*

## Pembersihan dan Pra-pemrosesan Data

Pembersihan data dilakukan dengan tujuan untuk memastikan kualitas data yang akan dipelajari oleh model LSTM. Pembersihan data dilakukan dengan berbagai cara, di antaranya adalah menyaring data pertandingan dengan *rating* pemain yang rendah, membuang data pertandingan yang terlalu pendek, dan membuang informasi dalam data yang tidak diperlukan oleh model. Berbagai teknik pra-pemrosesan data juga dilakukan, seperti konversi notasi gerakan dari notasi *Short Algrebatic Term* menjadi *Long Algrebaic Term* menggunakan alat bantu PGN-Extract, konversi format data dari .PGN menjadi .CSV, serta pembagian data menjadi data latih dan data validasi. Hal-hal tersebut dilakukan untuk mengubah dan mengolah bentuk data agar dapat diterima dengan baik oleh model LSTM.

## Pembangunan dan Pelatihan Model

Model LSTM akan dibangun untuk memprediksi gerakan terbaik untuk dimainkan pada suatu kondisi tertentu, berdasarkan rangkaian gerakan yang telah dilakukan. Pembangunan model LSTM akan dilakukan dengan menggunakan *library* Pytorch, dan pelatihan model akan dilakukan dengan menggunakan data latih yang telah dipersiapkan pada tahap sebelumnya. Tahap ini, bersama dengan tahap evaluasi dan optimasi model, akan dilakukan secara iteratif, hingga kinerja model dirasa memuaskan.

## Evaluasi dan Optimasi Model

Evaluasi model LSTM yang telah dibangun akan dilakukan dengan mempertimbangkan berbagai metrik evaluasi seperti *loss,* akurasi, *f1, precision,* dan *recall,* untuk menentukan baik atau buruknya kinerja model*.* Evaluasi juga dilakukan sebagai upaya untuk mengidentifikasi penyebab dari permasalahan terkait kinerja model, beserta dengan solusi perbaikannya. Optimasi model melibatkan pembaruan, penambahan, atau pengaturan ulang arsitektur model berdasarkan evaluasi yang telah dilakukan untuk mencapai kinerja model yang lebih baik.

## Pembangunan Mesin Catur

Pembangunan mesin catur akan dilakukan dengan menggunakan Python sebagai bahasa pemrograman, mengikuti protokol UCI atau *Universal Chess Interface* sebagai protokol komunikasi*,* dan mengimplementasikan model LSTM yang telah dibangun sebagai *move generator.* UCI adalah standar protokol komunikasi antara program GUI catur dengan mesin catur. Dengan mengimplementasikan protokol UCI dalam pembangunannya, mesin catur nantinya akan dapat dimainkan di berbagai antarmuka catur yang telah ada dan tersedia untuk digunakan seperti Arena Chess, Cute Chess, dan Lichess.

## Pengujian dan Analisis Hasil

Cara terbaik untuk menguji sebuah mesin catur adalah dengan menandingkannya, baik melawan manusia maupun melawan mesin catur lain. Pertandingan akan dilakukan dengan bantuan antarmuka catur Cute Chess, yang juga menyediakan fitur untuk melakukan turnamen otomatis antara dua atau lebih mesin catur. Hasil dari pertandingan tersebut akan direkap untuk melihat jumlah kemenangan, kekalahan, dan seri, serta dilihat akurasinya menggunakan alat bantu berupa fitur *analyze board* milik lichess.org. Akan dilakukan pula observasi terhadap pertandingan yang telah dilakukan untuk menganalisis karakteristik permainan dari mesin catur yang telah dibangun. Berdasarkan hasil analisis tersebut, dapat ditarik kesimpulan terkait tingkat keberhasilan dari penelitian ini.

## Peralatan Pendukung

### Perangkat Keras

Perangkat keras yang digunakan pada penelitian ini adalah laptop dengan spesifikasi sebagaimana dapat dilihat pada bagian selanjutnya.

1. Processor Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz.
2. *Random Access Memory* (RAM) 16GB.
3. *Solid State Drive* (SSD) 500GB.
4. *Graphical Processing Unit* (GPU)Nvidia GeForce GTX 1650.

### Perangkat Lunak

Perangkat lunak yang digunakan pada penelitian ini dapat dilihat pada bagian selanjutnya.

1. Sistem Operasi Windows 11 64-bit.
2. Bahasa pemrograman Python versi 3.10.
3. Halaman web DeepNote dan Google Colab sebagai lingkungan pemrograman.
4. *Library* Python NumPy, Pandas, PyTorch, dan Chess.
5. Perangkat lunak PGN-Extract dan Cute Chess

# PERANCANGAN

## Perancangan Algoritma

### Pembersihan Data

Tahap pembersihan data bertujuan untuk membersihkan data dari informasi-informasi yang tidak diinginkan seperti data yang rusak atau data yang tidak relevan. Pada Gambar 2.5 terlihat bahwa data mentah berupa fail .PGN mengandung berbagai informasi yang tidak dibutuhkan oleh model, seperti *username* pemain, tanggal pertandingan, dan *rating* pemain. Informasi-informasi tersebut akan dibuang dari data, menyisakan rangkaian gerakan pertandingan untuk dipelajari oleh model. Pengambilan rangkaian gerakan dibatasi pada pertandingan yang dimenangkan oleh putih dengan sekakmat, hal tersebut dilakukan agar model hanya mempelajari gerakan dari pihak yang menang (putih), dan memastikan bahwa rangkaian gerakan mengandung informasi yang utuh dari awal hingga akhir pertandingan.

Selanjutnya, rangkaian gerakan dipisahkan berdasarkan spasi, sehingga setiap gerakan memiliki 1 indeks. Data kemudian dibersihkan kembali dari pertandingan dengan jumlah gerakan yang lebih dari 100. Hal tersebut dilakukan karena pertandingan yang terlalu panjang cenderung mengandung gerakan-gerakan yang terlalu abstrak menjelang akhir pertandingan, dengan raja salah satu pihak hanya berlari-lari menghindari serangan lawan karena tidak tersisa buah lain untuk melindunginya. Berdasarkan uraian tersebut, maka rancangan algoritma untuk tahap pembersihan data dapat dilihat pada Kode Program 4.1.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | games ← Array 2D  **input** fail PGN  **for** setiap data pertandingan dalam games **do**  **if** putih menang dengan sekakmat **then**  **insert** rangkaian gerakan ke games  **end for**  **for** setiap baris dalam games **do**  **split** gerakan dalam baris berdasarkan “ “  **if** jumlah gerakan dalam baris > 100 **then**  **remove** baris  **end for** |

Kode Program 4.1 Kode Semu Pembersihan Data

Kode semu diawali dengan deklarasi variabel *games* berupa *array* 2D untuk menyimpan rangkaian gerakan, kemudian diterima masukan berupa fail .PGN. Selanjutnya, pada baris ke-3 sampai dengan ke-6 dilakukan perulangan melalui setiap data pertandingan dalam fail, jika pertandingan dimenangkan oleh putih dengan sekakmat, maka rangkaian pertandingan tersebut disisipkan ke variabel *games.* Selanjutnya, pada baris ke-7 sampai dengan ke-11, dilakukan perulangan melalui setiap baris dalam *games,* baris kemudian dipisahkan berdasarkan spasi, dan baris dengan jumlah gerakan yang lebih dari 100 akan dihapus dari *games.*

### Pra-pemrosesan Data

Tahap pra-pemrosesan data bertujuan untuk memodifikasi bentuk dan format data agar dapat diterima dengan baik oleh model. Dalam penelitian ini, notasi gerakan yang digunakan memiliki beberapa karakter yang tidak diperlukan oleh model, seperti karakter “x” yang menyatakan tangkapan, karakter “#” yang menyatakan sekakmat, dan karakter “+” yang menyatakan sekak. Karakter-karakter tersebut akan dibuang dari data, menyisakan inisial buah dan petak tujuan gerak. Terdapat 5 kondisi dengan gerakan yang perlu dibersihkan, sebagaimana dapat dilihat pada Tabel 4.1.

Tabel 4.1 Pembersihan Gerakan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Kondisi** | **Contoh Gerakan** | **Arti Gerakan** | **Gerakan setelah dibersihkan** |
| 1. | Sekak | “Qe5+” | Menteri ke e5, sekak | “Qe5” |
| 2. | Tangkapan | “Qxe5” | Menteri ke e5, menangkap buah lawan. | “Qe5” |
| 3. | Sekakmat | “Qe5#” | Menteri ke e5, sekakmat | “Qe5” |
| 4. | Promosi pion | “e8=Q” | Pion ke e8, promosi menjadi menteri | “e8” |
| 5. | Gerakan dengan petak awal | “Ne4g6” | Kuda dari e4 ke g6 | “Ng6” |

Selanjutnya, dilakukan pembuatan *dictionary* atau kamus dari seluruh gerakan berbeda yang ada dalam data. Kamus ini kemudian digunakan untuk mengkodekan setiap gerakan menjadi *integer,* dengan cara mengganti gerakan tersebut menjadi indeksnya pada kamus. Sebagai contoh, rangkaian gerakan [“e4”, “e5”, “Nf3”, “Nc6”, “d4”] akan membentuk kamus {1:“e4”, 2:“e5”, 3:“Nf3”, 4: “Nc6”, 5: “d4”}, sehingga rangkaian tersebut akan dikodekan menjadi [1, 2, 3, 4, 5]. Selanjutnya, dilakukan pembuatan *N-grams* dari rangkaian gerakan, yang memecah setiap rangkaian gerakan dengan panjang N menjadi buah rangkaian, dengan panjang masing-masing rangkaian adalah . Sebagai contoh, rangkaian gerakan [1, 2, 3, 4, 5] yang memiliki panjang 5, akan dipecah menjadi 2 buah rangkaian dengan panjang 3, dan 5, yaitu:

* [1, 2, 3]
* [1, 2, 3, 4, 5]

Selanjutnya, *N-grams* diberikan *pre-padding* berupa angka 0, untuk menyeragamkan panjang dari seluruh baris *N-grams*. Sebagai contoh, 2 baris *N-grams* sebelumnya akan disisipkan angka 0 di bagian awal, sehingga menjadi:

* [0, 0, 1, 2, 3]
* [1, 2, 3, 4, 5]

Terakhir, setiap baris data akan dipisahkan menjadi fitur dan label, dengan mengambil indeks terakhir dari baris dan memisahkannya menjadi label. Sebagai contoh, 3 baris data sebelumnya akan dipisahkan sebagaimana pada Tabel 4.1.

Tabel 4.2 Pemisahan Fitur dan Label

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Baris Data Awal** | **Fitur** | **Label** |
| 1. | [0, 0, 1, 2, 3] | [0, 0, 1, 2] | 3 |
| 2. | [1, 2, 3, 4, 5] | [1, 2, 3, 4] | 5 |

Berdasarkan uraian sebelumnya, rancangan algoritma dari tahap pra-pemrosesan data dapat dilihat pada Kode Program 4.2.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | dict ←Array 1D, seq ←Array 2D, fitur ← Array 2D,  label ← Array 1D, M ← jumlah gerakan pada baris terpanjang,  **input** games  **for** setiap baris dalam games **do**  **for** setiap gerakan dalam baris **do**  **remove** karakter yang tidak diinginkan  **if** gerakan tidak ada dalam dict **then**  **insert** gerakan ke dict  **replace** gerakan dengan indeks-nya di dict  **end for**  **for** setiap baris dalam games **do**  N ← panjang baris  **for** j ← 0 **to** (N - 1) **do**  **insert** elemen baris pada index 0 sampai j+2 ke seq  **end for**  **for** setiap baris dalam seq **do**  N ← panjang baris  **for** j ← 0 **to** (M - N) **do**  **insert** 0 di awal baris  **end for**  **for** setiap baris dalam seq **do**  **insert** elemen baris pada index 0 sampai (M-1) ke fitur  **insert** elemen baris pada index M ke label  **end for** |

Kode Program 4.2 Kode Semu Pra-pemrosesan Data

Kode semu diawali dengan deklarasi variabel-variabel yang dibutuhkan. Variabel *dict* berupa *array* 1D akan digunakan sebagai kamus gerakan, variabel *seq* berupa array 2D akan digunakan untuk menyimpan *N-grams*, variabel *fitur* berupa array 1D akan digunakan untuk menyimpan fitur yang telah dipisahkan dari *N-grams*, variabel *label* berupa *array* 1D akan digunakan untuk menyimpan label yang telah dipisahkan dari *N-grams*, dan variabel *M* akan digunakan untuk menyimpan jumlah gerakan pada baris terpanjang.

Pengkodean data menjadi *integer* dilakukan pada baris ke-3 sampai dengan baris ke-9. Diawali dengan menerima variabel *games* berisi rangkaian gerakan yang telah dibersihkan pada tahap sebelumnya, kemudian dilakukan perulangan bersarang melalui setiap baris dalam *games* dan setiap gerakan dalam baris. Setiap gerakan kemudian dibersihkan dari karakter yang tidak diinginkan, kemudian disisipkan ke variabel *dict* jika gerakan tersebut belum pernah ditemukan sebelumnya. Gerakan dalam baris kemudian ditukar dengan indeks gerakan tersebut di variabel *dict.*

Pembuatan *N-grams* dilakukan pada baris ke-10 sampai dengan baris ke-14. Diawali dengan melakukan perulangan melalui setiap baris dalam *games*, kemudian menghitung panjang baris dan menyimpannya pada variabel *N*. Selanjutnya, dilakukan perulangan melalui inkremen *j*=0 hingga *j*=(*N*-1). Pada setiap nilai *j*, disisipkan gerakan pada indeks ke-0 hingga indeks ke-(*j*+2) ke variabel *seq*.

Pemberian *pre-padding* dilakukan pada baris ke-15 sampai dengan baris ke-19 dengan melakukan perulangan melalui setiap baris dalam *seq*. Pada setiap baris, dilakukan pengambilan panjang baris yang disimpan pada variabel *N*, kemudian dilakukan perulangan melalui inkremen *j*=0 hingga *j*=*M*-*N.* Pada setiap nilai *j*, disisipkan angka 0 ke awal baris.

Terakhir, pemisahan fitur dan label dilakukan pada baris ke-20 hingga baris ke-23, dengan melakukan perulangan melalui setiap baris dalam *seq*. Indeks terakhir pada baris kemudian disisipkan ke variabel *label,* sedangkan yang lainnya disisipkan ke variabel *fitur.*

### Kelas Model

Kelas model berisi atribut dan *method* milik objek model. Pada penelitian ini, objek model memiliki atribut lapisan *embedding*, lapisan LSTM, dan lapisan linear. Terdapat 2 *method* dari kelas ini, yaitu *method* konstruktor Model() untuk inisialisasi objek model, dan method predict() untuk memprediksi sebuah data menggunakan objek model. Rancangan algoritma dari kelas model dapat dilihat pada Kode Program 4.4.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | **class** Model  atribut embedLayer  atribut lstmLayer  atribut linearLayer  **constructor** Model(embedDim, hiddenSize, numClass):  embedLayer ← lapisan embedding dengan dimensi = embedDim  lstmLayer ← lapisan LSTM dengan jumlah hidden = hiddenSize  linearLayer ← lapisan linear dengan jumlah neuron = numClass  **method** predict(data):  output\_embed ← proses data melalui embedding\_layer  output\_lstm ← proses output\_embed melalui lstm\_layer  output\_model ← proses output\_lstm melalui linear\_layer  **return** output\_model  **end class** |

Kode Program 4.3 Kode Semu Kelas Model

Kode semu diawali dengan deklarasi kelas model pada baris pertama, kemudian pada baris ke-2 sampai dengan baris ke-4 dilakukan deklarasi atribut *embedLayer* berupa lapisan *embedding*, *lstmLayer* berupa lapisan LSTM, dan *linearLayer* berupa lapisan linear atau lapisan *dense*. Selanjutnya, pada baris ke-5 sampai dengan baris ke-8 dilakukan implementasi *method* konstruktor dengan parameter *embedDim* berupa *integer* untuk menentukan dimensi dari *embedLayer*, parameter *hiddenSize* berupa *integer* untuk menentukan jumlah *hidden unit* dari *lstmLayer*, dan parameter *numClass* berupa *integer* untuk menentukan jumlah neuron dari *linearLayer*.

Terakhir, pada baris ke-9 sampai dengan baris ke-13 dilakukan deklarasi *method* predict() dengan parameter *data* sebagai data yang ingin diprediksi, di dalam *method*, data diproses oleh setiap lapisan secara berurutan hingga didapatkan keluaran dari *linearLayer*, yang kemudian dikembalikan sebagai keluaran model.

### Pelatihan Model

Tahap pelatihan model diawali dengan melakukan inisialisasi objek model, kemudian melatih model tersebut sebanyak jumlah *epoch* yang telah ditentukan. Pada akhir setiap *epoch*, dilakukan pengujian model dengan memprediksi data uji dan menghitung *Average Centipawn Loss* dari hasil prediksi. Selanjutnya, setelah semua *epoch* selesai dilakukan, maka model akan disimpan sebagai fail .pt. Rancangan algoritma dari tahap ini dapat dilihat pada Kode Program 4.5.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | input\_size ← \_, hidden\_size ←\_, , num\_classes ← \_  model ← Model(input\_size, hidden\_size, num\_layers, num\_classes)  engine ← stockfish.exe  **for** i=0 **to** i=14 **do**  **for** setiap baris dalam fitur dan label **do**  output ← model.predict(fitur)  loss ← hitung loss antara output dan label  lakukan propagasi mundur  perbarui bobot berdasarkan propagasi mundur  **end for**  **output** loss  avgCploss ← 0  **for** setiap fitur uji dan label uji dalam data uji **do**  logits ← model(fitur)  output ← hasil pasca-pemrosesan logits  cpBefore ← hitung centipawn fitur uji menggunakan engine  cpAfter ← hitung centipawn output menggunakan engine  avgCploss ← avgCploss + (cpBefore - cpAfter)  avgCploss ← (avgCploss / panjang dataUji)  **end for**  **output** avgCploss  **end for**  **save** model sebagai modellstm.pt |

Kode Program 4.4 Kode Semu Pelatihan dan Pengujian Model

Kode semu diawali dengan menentukan *hyperparameter* yang dibutuhkan, yaitu *input\_size* sebagai panjang data yang diterima model LSTM, *hidden\_size* sebagai jumlah *hidden unit*, dan *num\_class* sebagai jumlah kelas dalam data sekaligus jumlah neuron pada lapisan keluaran. Selanjutnya, dilakukan inisialisasi objek model dengan memanggil konstruktor Model() dengan parameter yang telah ditentukan sebelumnya. Selanjutnya, pada baris ke-3 Stockfish dimuat ke variabel *engine* untuk menghitung *centipawn score* saat tahap pengujian.

Pelatihan model dilakukan pada baris ke-4 sampai dengan baris ke-9. Diawali dengan melakukan perulangan sebanyak jumlah *epoch* yang diinginkan, untuk setiap *epoch* dilakukan prediksi setiap data latih, perhitungan loss antara hasil prediksi dengan label, dan pembaruan bobot model dengan menerapkanpropagasi mundur. Pengujian model dilakukan di akhir setiap epoch, pada baris ke-12 sampai dengan baris ke-18. Diawali dengan memprediksi setiap data uji, kemudian melakukan pasca-pemrosesan untuk mendapatkan hasil prediksi yang valid, tahapan dari pasca-pemrosesan akan dijelaskan pada Subbab berikutnya. Selanjutnya, dilakukan perhitungan nilai *centipawn loss* dari setiap hasil prediksi mengunakan *engine*. *Centipawn loss* didapatkan dengan cara menghitung selisih antara nilai *centipawn score* fitur dan nilai *centipawn score* hasil prediksi. *Centipawn loss* dari setiap data uji kemudian dijumlahkan dan dibagi dengan banyak data uji, sehingga didapatkan nilai *Average Centipawn Loss,* yaitu rata-rata *centipawn loss* dari seluruh data uji*.*

#### Pasca-pemrosesan

Pasca-pemrosesan dilakukan untuk menangani hasil prediksi yang tidak dapat dimainkan oleh mesin catur. Terdapat 3 kondisi yang akan ditangani oleh pasca-pemrosesan, yaitu gerakan ambigu, promosi pion, dan gerakan ilegal, sebagaimana dapat dilihat pada Tabel 4.3.

Tabel 4.3 Pasca-pemrosesan Gerakan

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Keterangan contoh kondisi** | **Gerakan awal** | **Penanganan** |
| 1. | Terdapat kuda pada c3 dan e3, keduanya dapat bergerak ke d5 | Nd5 | Ubah gerakan menjadi Nc3d5 atau Ne3d5, dipilih secara acak |
| 2. | Promosi pion pada petak e8 | e8 | Ubah gerakan menjadi e8=Q |
| 3. | gerakan ilegal karena ratu telah tertangkap | Qe5 | Temukan *logit* terbesar yang legal untuk dimainkan |

Gerakan ambigu akan ditangani dengan menambahkan petak awal salah satu buah yang dipilih secara acak setelah huruf pertama gerakan. Promosi pion akan ditangani dengan menambahkan *string* “=Q” di akhir gerakan, sehingga pion akan selalu dipromosikan menjadi menteri yang merupakan buah terkuat. Gerakan ilegal akan ditangani dengan mencari logit terbesar yang legal untuk dimainkan. Berdasarkan penjelasan tersebut, rancangan algoritma dari pasca-pemrosesan dapat dilihat pada Kode Program 4.5.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | logits ← keluaran model  move ← kelas dengan nilai logits tertinggi  **if** move illegal untuk dimainkan **then**  max ←  **for** setiap logit dalam logits **do**  **if** indeks logit = 0 **then**  **continue**  **if** kelas logit legal **and** logit > max **then**  max ← logit  move ← kelas logit  **end for**  **elif** move merupakan gerakan ambigu **then**  sisipkan petak awal salah satu buah setelah huruf pertama move  **elif** move merupakan promosi pion **then**  sisipkan “=Q” di akhir move |

Kode Program 4.5 Kode Semu Pasca-pemrosesan

### Mesin Catur

Mesin catur akan dibangun dengan mengimplementasikan protokol komunikasi *Universal Chess Interface* atau UCI*.* Terdapat 5 perintah utama dalam protokol UCI yang harus diimplementasikan saat membangun mesin catur agar mesin catur dapat berjalan dengan baik, yaitu:

1. perintah “uci” yang mengharapkan respon “uciok” dari mesin catur.
2. perintah “position startpos“ yang menyatakan posisi papan terkini untuk diprediksi oleh mesin catur.
3. perintah “isready” yang mengharapkan respon “readyok” dari mesin catur.
4. perintah “go” yang mengharapkan respon berupa gerakan yang ingin dimainkan oleh mesin catur, diawali dengan kata kunci “bestmove”.
5. Perintah “quit” yang digunakan untuk menghentikan kerja sistem

Rancangan algoritmadari mesin catur yang mengimplementasikan 5 perintah tersebut dapat dilihat pada Kode Program 4.6.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | model ← modellstm.pt  **while** continue = True **do**  **input** msg  **if** msg = "uci" **then**  **output** "Licheese by Rafi Indra Fattah\nuciok"  **else if** msg = "isready" **then**  **output** "readyok"  **else if** msg diawali dengan "position startpos" **then**  moves ← kata ke-4 sampai dengan kata terakhir dalam msg  logits ← model.predict(moves)  nextMove ← hasil pasca pemrosesan logits  **else if** msg diawali dengan "go" **then**  **output** “bestmove ” + nextMove  **else if** msg = "quit" **then**  continue = False |

Kode Program 4.6 Kode Semu Mesin Catur

Kode semu diawali dengan memuat model LSTM yang telah dilatih ke variabel model, dan deklarasi variabel *continue* berupa boolean dengan nilai awal True yang digunakan untuk menghentikan kerja sistem. Selanjutnya, dilakukan perulangan yang akan terus berlanjut selama *continue* bernilai True, untuk setiap iterasi, diterima masukan berupa perintah UCI yang disimpan pada variabel *msg,* nilai *msg* akan menentukan aksi apa yang akan dilakukan oleh mesin catur.

Perintah “uci” diimplementasikan pada baris ke-4 dan baris ke-5, dengan melakukan seleksi kondisi berupa *msg* bernilai “uci”, jika seleksi kondisi terpenuhi, maka mesin catur akan menampilkan teks “Licheese by Rafi Indra Fattah”, diikuti dengan teks “uciok” pada baris berikutnya.

Perintah “isready” diimplementasikan pada baris ke-6 dan baris ke-7 dengan melakukan seleksi kondisi berupa *msg* bernilai “isready, jika seleksi kondisi terpenuhi, maka mesin catur akan menampilkan teks “readyok”

Perintah “position startpos” diimplementasikan pada baris ke-8 sampai dengan baris ke-11 dengan melakukan seleksi kondisi jika *msg* diawali dengan “position startpos”, jika seleksi kondisi terpenuhi, maka mesin catur akan mengambil kata ke-4 sampai dengan kata terakhir pada *msg* dan menyimpannya pada variabel *moves.* Mesin catur kemudian melakukan prediksi *moves* menggunakan model dan menyimpan keluaran model pada variabel *logits.* Terakhir, kelas dengan nilai logits tertinggi akan disimpan pada variabel *nextMove* sebagai hasil prediksi.

Perintah “go” diimplementasikan pada baris ke-12 dan baris ke-13 dengan melakukan seleksi kondisi jika *msg* diawali dengan kata “go”, jika seleksi kodisi terpenuhi, maka mesin catur akan menampilkan teks “bestmove ” yang diikuti oleh variabel *nextMove* pada kata berikutnya.

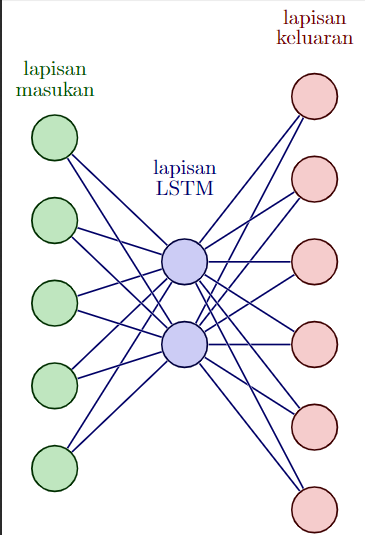
Terakhir, perintah “quit” diimplementasikan pada baris ke-14 dan baris ke-15, dengan melakukan seleksi kondisi jika *msg* bernilai “quit”, jika seleksi kondisi terpenuhi, maka dilakukan pengubahan nilai variabel *continue* dari True menjadi False, sehingga menghentikan perulangan pada baris ke-2.

## Perhitungan Manual

Subbab ini melakukan perhitungan untuk pelatihan model sebanyak 1 *epoch* terhadap 1 baris data latih, dan melakukan prediksi 1 baris data uji. Data latih dan data uji yang digunakan dapat dilihat pada subbab selanjutnya. Rincian arsitektur model yang digunakan pada perhitungan manual ini dapat dilihat pada Tabel 4.2, sedangkan visualisasi arsitektur model dapat dilihat pada gambar 4.1. Arsitektur tersebut dipilih untuk mempermudah perhitungan pada subbab ini, dan tidak merepresentasikan arsitektur akhir model yang digunakan pada penelitian ini.

Tabel 4.4 Arsitektur Model untuk Perhitungan Manual

|  |  |  |
| --- | --- | --- |
| No. | **Jenis Lapisan** | ***Hyperparameter*** |
| 1. | Lapisan *Input* | Jumlah neuron = 4 |
| 2. | Lapisan LSTM | Jumlah *hidden unit* = 2 |
| 3. | Lapisan *linear* | Jumlah *neuron* = 6 |



Gambar 4.1 Arsitektur Model untuk Perhitungan Manual

### Pra-pemrosesan

Subbab ini melakukan pra-pemrosesan pada 1 contoh baris data. Bentuk data sebelum dan sesudah dilakukan pra-pemrosesan dapat dilihat pada Tabel 4.5.

Tabel 4.5 Pra-pemrosesan untuk Perhitungan Manual

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tahap** | **No.** | **Nilai** | | |
| Data Awal | 1. | [“e4”, “e5”, “Nf3”, “Nc6”, “d4”] | | |
| Kamus kelas yang terbentuk | 1. | 1: “e4” | | |
| 2. | 2: “e5” | | |
| 3. | 3: “Nf3” | | |
| 4. | 4: “Nc6” | | |
| 5. | 5: “d4” | | |
| Setelah dikodekan menjadi *integer* | 1. | [1, 2, 3, 4, 5] | | |
| Setelah dibentuk *N-Grams* | 1. | [1, 2, 3] | | |
| 2. | [1, 2, 3, 4, 5] | | |
| Setelah diberikan *padding* | 1. | [0, 0, 1, 2, 3] | | |
| 2. | [1, 2, 3, 4, 5] | | |
| Setelah Pemisahan fitur dan label |  | **Jenis Data** | **Fitur** | **Label** |
| 1. | Data latih | [0, 0, 1, 2] | 3 |
| 2. | Data uji | [1, 2, 3, 4] | 5 |

Contoh baris data yang digunakan membentuk 2 baris fitur dan label setelah dilakukan pra-pemrosesan. Pada perhitungan selanjutnya, fitur dan label pertama akan digunakan sebagai data latih, sedangkan fitur dan label ke-2 akan digunakan sebagai data uji.

### Inisialisasi Bobot

Inisialisasi yang perlu dilakukan untuk lapisan LSTM meliputi bobot awal masing-masing *gate,* nilai awal *hidden state,* dan nilai awal *cell state*. Dimensi dari bobot masing-masing *gate* pada lapisan LSTM adalah (*hidden size* xpanjang data)*,* menyesuaikan dengan bentuk data dan *hidden size* yang digunakan pada perhitungan manual ini, maka dimensi bobot yang digunakan adalah (2x4). Bobot awal untuk setiap *gate* diinisialisasikan oleh model LSTM secara acak, sebagaimana dapat dilihat pada Tabel 4.6.

Tabel 4.6 Inisialisasi Bobot *Gate* LSTM

|  |  |
| --- | --- |
| **Jenis *Gate*** | **Bobot Awal** |
| *Forget Gate* |  |
| *Input Gate* |  |
| *Candidate gate* |  |
| *Output Gate* |  |

Panjang dari *hidden state* dan *cell state* ditentukan oleh jumlah *hidden unit*. Menyesuaikan dengan *hidden unit* yang digunakan pada perhitungan manual ini, maka dimensi dari *hidden state* dan *cell state* adalah (1x2), dengan nilai awal diinisialisasikan sebagai 0, dan bobot awal dengan dimensi (2x2) yang diinisialisasikan sebagai 1, sebagaimana dapat dilihat pada Tabel 4.7.

Tabel 4.7 Inisialisasi *Hidden State* dan *Cell State* Awal

|  |  |  |
| --- | --- | --- |
| **Jenis State** | **Nilai Awal** | **Bobot Awal** |
| *Hidden State* | [[0, 0]] | [[1, 1], [1, 1]] |
| *Cell State* | [[0, 0]] | [[1, 1], [1, 1]] |

Inisialisasi untuk lapisanlinear meliputi semua bobot yang menghubungkan lapisan LSTM dan lapisan linear. Terdapat 2 neuron pada lapisan LSTM, sehingga setiap neuron pada lapisan linear memiliki 2 bobot pula. Inisialisasi bobot lapisan linear juga dilakukan secara acak, sebagaimana dapat dilihat pada Tabel 4.8.

Tabel 4.8 Inisialisasi Bobot Awal Lapisan Linear

|  |  |
| --- | --- |
| **Neuron** | **Bobot Awal** |
| 1 | [-0,8139, -0,2747] |
| 2 | [-0.4778, -1,4873] |
| 3 | [1,4972, -0,997] |
| 4 | [-2,9437, -1,8984] |
| 5 | [0,6662, -0,9028] |
| 6 | [-1,3079, 0,6734] |

### Pelatihan Model

#### Propagasi Maju

Sebagaimana telah dijelaskan sebelumnya, pelatihan model akan dilakukan sebanyak 1 *epoch* terhadap fitur dengan nilai [0,0,1,2] dan label dengan nilai 3. Tahap pertama dari pelatihan model adalah propagasi maju. Mengacu pada Persamaan 2.3 sampai dengan Persamaan 2.8 untuk melakukan propagasi maju pada lapisan LSTM, maka didapatkan perhitungan sebagaimana dapat dilihat pada bagian selanjutnya.

Perhitungan *forget gate*

Perhitungan *candidate gate*

Perhitungan *input gate*

Perhitungan *cell state baru*

Perhitungan *output gate*

Perhitungan *hidden state baru*

Berdasarkan perhitungan sebelumnya, didapatkan keluaran lapisan LSTM dengan nilai , berikutnya nilai tersebut digunakan sebagai masukan lapisan linear untuk diproses menjadi nilai *logits* setiap kelas. Mengacu pada Persamaan 2.16 untuk melakukan propagasi maju pada lapisan linear, maka didapatkan perhitungan sebagaimana dapat dilihat pada bagian selanjutnya.

Perhitungan lapisan linear

#### Perhitungan loss

Perhitungan *loss* dilakukan dengan menggunakan fungsi *loss crossentropy.* Mengacu pada Persamaan 2.21 untuk menghitung nilai *loss crossentropy,* maka didapatkan perhitungan sebagaimana dapat dilihat pada bagian selanjutnya.

#### Propagasi Mundur

Tahap berikutnya dari pelatihan model adalah propagasi mundur untuk memperbarui bobot model. Propagasi mundur dilakukan dari lapisan terluar model, yaitu lapisan linear. Mengacu pada Persamaan 2.18 dan 2.19 untuk melakukan propagasi mundur pada lapisan linear, maka didapatkan perhitungan sebagaimana dapat dilihat pada bagian selanjutnya.

Perhitungan gradien lapisan linear

Perhitungan bobot baru lapisan linear

Perhitungan sebelumnya menggunakan *learning rate* () dengan nilai 0,1, sehingga didapatkan bobot baru untuk lapisan linear*,* sebagaimana dapat dilihat pada Tabel 4.9.

Tabel 4.9 Bobot Baru Lapisan Linear

|  |  |
| --- | --- |
| **Neuron** | **Bobot** |
| 1 | [-0,7339, -0,2337] |
| 2 | [-0.3936, -1.4441] |
| 3 | [1,534, 1,0159] |
| 4 | [-2,8205, -1,8352] |
| 5 | [0,7293, -0,8704] |
| 6 | [-1,2278, 0,7145] |

Berikutnya, dilakukan propagasi mundur untuk memperbarui bobot masing-masing *gate* pada lapisan LSTM. Mengacu pada Persamaan 2.9 sampai dengan Persamaan dan 2.15 untuk melakukan propagasi mundur pada lapisan LSTM, maka didapatkan perhitungan sebagaimana dapat dilihat pada bagian selanjutnya.

Perhitungan turunan *loss* terhadap x

Perhitungan turunan loss terhadap *hidden state*

Perhitungan turunan loss terhadap *cell state*

Perhitungan turunan loss terhadap *candidate gate*

Perhitungan turunan loss terhadap *input gate*

Perhitungan turunan loss terhadap *forget gate*

Perhitungan turunan loss terhadap *output gate*

Perhitungan gradien *candidate gate*

Perhitungan gradien *input gate*

Perhitungan gradien *forget gate*

Perhitungan gradien *output gate*

Perhitungan bobot baru *candidate gate*

Perhitungan bobot baru *input gate*

Perhitungan bobot baru *forget gate*

Perhitungan bobot baru *output gate*

Perhitungan manual sebelumnya menggunakan *learning rate* () dengan nilai 0.1, sehingga didapatkan bobot baru untuk masing-masing *gate* pada lapisan LSTM*,* sebagaimana dapat dilihat pada Tabel 4.10.

Tabel 4.10 Bobot Baru *Gate* Setelah Propagasi Mundur

|  |  |
| --- | --- |
| **Jenis *Gate*** | **Bobot Baru** |
| *Forget Gate* |  |
| *Input Gate* |  |
| *Candidate gate* |  |
| *Output Gate* |  |

Didapatkannya bobot baru model menandakan bahwa 1 *epoch* pelatihan model telah selesai dilakukan, jika ingin memulai *epoch* berikutnya, maka perhitungan akan dimulai kembali pada tahap pertama propagasi maju menggunakan bobot yang telah diperbarui, hingga tahap ke-17 propagasi mundur.

### Prediksi Data Uji

Model yang telah dilatih dapat digunakan untuk memprediksi data baru. Pada tahap ini, data uji yang digunakan memiliki fitur dengan nilai [1, 2, 3, 4], dan label dengan nilai 5, sebagaimana dapat dilihat pada Tabel 4.3. Tahap ini pada dasarnya melakukan propagasi maju terhadap data baru menggunakan bobot model yang telah diperbarui pada tahap pelatihan sebelumnya, sebagaimana dapat dilihat pada bagian selanjutnya.

Perhitungan *forget gate*

Perhitungan *candidate gate*

Perhitungan *input gate*

Perhitungan *cell state baru*

Perhitungan *output gate*

Perhitungan *hidden state baru*

Perhitungan lapisan linear

#### Pasca-pemrosesan

Sebelum mendapatkan hasil prediksi model, *logits* hasil keluaran model perlu dilakukan pasca-pemrosesan terlebih dahulu, untuk mengantisipasi kondisi saat nilai tertinggi dari *logits* merupakan gerakan yang tidak dapat dimainkan oleh mesin catur. Pasca-pemrosesan dilakukan sebagaimana dapat dilihat pada Tabel 4.11.

Tabel 4.11 Pasca-pemrosesan Perhitungan Manual

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Logit* yang diperiksa** | **Nilai** | **Indeks *logit*** | **Nilai kelas** | **Hasil pemeriksaan** |
| *Logit* terbesar | 0,8978 | 2 | e5 | Ilegal, periksa *logit* ke-1, inisialisasi *max* sebagai |
| *Logit* ke-1 | 0,076 | 1 | e4 | Ilegal, periksa *logit* ke-2 |
| *Logit* ke-2 | 0,8978 | 2 | e5 | Ilegal, periksa *logit* ke-3 |
| *Logit* ke-3 | -1,6592 | 3 | Nf6 | Ilegal, periksa *logit* ke-4 |
| *Logit* ke-4 | 0,7772 | 4 | Nc6 | Ilegal, periksa *logit* ke-5 |
| *Logit* ke-5 | -1,114 | 5 | d4 | Legal dan > *max*, gunakan sebagai hasil prediksi |

Berdasarkan pasca-pemrosesan tersebut, didapatkan bahwa hasil prediksi model adalah *logit* pada indeks ke-5 dengan nilai -1,114, menyatakan bahwa hasil prediksi model adalah gerakan pada kelas ke-5 dalam kamus, yaitu “d4”. Hasil prediksi tersebut sama dengan label, namun hal tersebut dikarenakan gerakan pada kelas ke-5 adalah satu-satunya gerakan dalam kamus yang dapat dimainkan dari posisi tersebut. Perbandingan antara nilai label dengan hasil prediksi dapat dilihat pada Tabel 4.12.

Tabel 4.12 Perbandingan Nilai Sesungguhnya dan Hasil Prediksi

|  |  |  |
| --- | --- | --- |
|  | **Nilai label** | **Hasil Prediksi** |
| **Indeks Kelas** | 4 | 4 |
| **Nilai Kelas** | “d4” | “d4” |

#### Perhitungan Loss

Berbeda dengan tahap pelatihan model yang menggunakan fungsi *loss crossentropy* untuk mengukur kinerja model*,* tahap pengujian model atau prediksi data uji akan menggunakan fungsi *Average Centipawn Loss* atau ACPLyang dihitung menggunakan bantuan Stockfish sebagai fungsi *loss,* sebagaimana telah dijelaskan pada Subbab 4.1.4. Fungsi *loss* ACPL tidak lagi mengacu kepada label dari data uji untuk menghitung *loss*, melainkan menghitung rata-rata perubahan nilai *centipawn score* setelah model melakukan prediksi gerakan, menghasilkan nilai *loss* yang lebih merepresentasikan kinerja model saat bermain catur, daripada performa model dalam memprediksi data uji.

## Rancangan Pengujian

### *Hyperparameter Tuning*

*Hyperparameter tuning* atau penyetelan *hyperparameter* dilakukan untuk menemukan nilai-nilai *hyperparameter* yang menghasilkan kinerja model terbaik. Pada penelitian ini, terdapat 3 buah *hyperparameter* yang akan disetel, yaitu *batch size* yang menentukan berapa banyak baris data yang akan diproses secara bersamaan, *hidden size* yang menentukan jumlah *hidden unit* dari lapisan LSTM, *embedding dimension* yang menentukan dimensi dari lapisan *embedding,* dan *num layers* yang menentukan jumlah lapisan LSTM dari model. Nilai-nilai yang akan diuji untuk masing-masing *hyperparameter* dapat dilihat pada Tabel 4.13.

Tabel 4.13 Nilai *hyperparameter*

|  |  |  |
| --- | --- | --- |
| **No.** | ***hyperparameter*** | **Nilai yang akan diuji** |
| 1 | *Batch size* | 250, 500, 1000 |
| 2 | *Embedding dimension* | 32, 64, 128 |
| 3 | *Hidden size* | 32, 64, 128 |
| 4 | *Num layers* | 1, 2 |

Penyetelan akan dilakukan dengan mencoba kombinasi antara nilai-nilai *hyperparameter* yang akan diuji dengan arsitektur dasar model. Arsitektur dasar model ditentukan dengan mengambil nilai tengah dari nilai-nilai *hyperparameter* yang terdapat pada Tabel 4.14. Berdasarkan uraian tersebut, terbentuk 11 buah skenario dari *hyperparameter tuning*, sebagaimanadapat dilihat pada Tabel 4.14.

Tabel 4.15 Rancangan *Hyperparameter Tuning*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Nama Skenario** | **No. Skenario** | ***Batch size*** | ***Hidden size*** | ***embedding dimension*** | ***Num layers*** |
| *Batch size* | 1. | 250 | 64 | 64 | 1 |
| 2. | 500 |
| 3. | 1000 |
| *Hidden size* | 4. | 500 | 32 | 64 | 1 |
| 5. | 64 |
| 6. | 128 |
| *Embedding dimension* | 7. | 500 | 64 | 32 | 1 |
| 8. | 64 |
| 9. | 128 |
| *Num layers* | 10. | 500 | 64 | 64 | 1 |
| 11. | 2 |

Setiap arsitektur pada Tabel 4.14 akan dilatih sebanyak 2 *epoch* terhadap sampel data latih sebanyak 200.000 baris data. Arsitektur yang menghasilkan nilai ACPL terendah kemudian akan digunakan sebagai arsitektur akhir model, untuk dilatih kembali menggunakan seluruh data latih pada tahap pelatihan dan pengujian model.

### Pengujian Melawan Mesin Catur Lain

Pengujian melawan mesin catur lain akan dilakukan melawan mesin catur dari berbagai tingkat keahlian yang dilihat berdasarkan estimasi *rating* dari mesin catur tersebut. Aspek yang akan dilihat untuk menentukan baik atau buruknya kinerja mesin catur yang telah dibangun adalah jumlah menang, jumlah kalah, jumlah seri, akurasi tertinggi, akurasi terendah, dan rata-rata akurasi. Rancangan dari pengujian melawan mesin catur lain dapat dilihat pada Tabel 4.14.

Tabel 4.16 Rancangan Pengujian Melawan Mesin Catur Lain

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **nama lawan** | **estimasi *rating* lawan** | **jumlah menang** | **jumlah kalah** | **jumlah seri** | **akurasi tertinggi** | **Akurasi terendah** | **rata-rata akurasi** |
|  | <500 |  |  |  |  |  |  |
|  | ~1000 |  |  |  |  |  |  |
|  | >1500 |  |  |  |  |  |  |

# IMPLEMENTASI

## Implementasi Algoritma

### Pembersihan Data

Tahap pertama dari pembersihan data adalah mengambil rangkaian gerakan dari pertandingan yang berakhir dengan sekakmat oleh putih, hal ini dilakukan dengan bantuan perangkat lunak PGN-Extract, sebagaimana dapat dilihat pada Kode Program 5.1.

|  |  |
| --- | --- |
| 1  2 | ./pgn-extract.exe games.pgn -Tr1-0 -C -D -M --notags --output huft.pgn |

Kode Program 5.1 Implementasi Pengambilan Rangkaian Gerakan

Kode program tersebut merupakan perintah kepada perangkat lunak PGN-Extract untuk membaca fail “games.pgn”, kemudian memilah dan menyimpan pertandingan yang memenuhi *flag* sebagai berikut:

1. -Tr1-0, menyatakan pertandingan yang dimenangkan oleh putih.
2. -M, menyatakan pertandingan yang berakhir dengan sekakmat.
3. -C, menyatakan untuk membuang semua komentar dalam data.
4. -D, menyatakan untuk membuang data duplikat.
5. --notags, menyatakan untuk membuang semua informasi selain rangkaian gerakan seperti nama pemain, *rating* pemain, dan tanggal pertandingan.
6. –output huft.txt, menyatakan untuk menyimpan hasil ke dalam fail “huft.txt”.

Tahap berikutnya dari pembersihan data adalah membuang pertandingan yang memiliki jumlah gerakan > 100, hal ini dilakukan menggunakan bahasa pemrograman Python, sebagaimana dapat dilihat pada kode program 5.2.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | init  = []  temp = ""  with open("WTF.txt", "r") as file:    for line in file:      if (line.split()):        temp+=line.replace("$2","").replace("$4","").replace("$6","")      else:        init.append(temp.split()[:-1])        temp = ""  file.close()  cleaned = [game for game in init if len(game) < 100] |

Kode Program 5.2 Implementasi Pembersihan Data

Penjelasan untuk masing-masing baris pada Kode Program 5.2 adalah sebagai berikut:

Baris ke-1 merupakan deklarasi variabel *init* berupa array 1 dimensi untuk menyimpan baris data pada fail yang akan dibaca

Baris ke-2 merupakan deklarasi variabel *temp* berupa *string* kosong untuk menampung baris data fail yang sedang dibaca secara sementara, sebelum disisipkan ke *init.*

Baris ke-3 membuka fail “WTF.txt” sebagai variabel *file.*

Baris ke-4 membaca setiap baris dalam *file* sebagai variabel *line.*

Baris ke 5 melakukan seleksi kondisi jika *line* bukan merupakan *string* kosong.

Baris ke-6 menggabungkan *line* ke *temp* jika seleksi kondisi pada baris ke-5 terpenuhi, sekaligus menghapus *string* “$2”, “$4”, dan “$6” yang merupakan kesalahan dalam penulisan data.

Baris ke-7 merupakan *statement else* dari seleksi kondisi pada baris ke-5.

Baris ke-8 memisahkan *temp* berdasarkan spasi dan membuang kata terakhir, kemudian menambahkan *temp* ke *init,* jika seleksi kondisi pada baris ke-5 tidak terpenuhi.

Baris ke-9 mengembalikan nilai *temp* menjadi *string* kosong.

Baris ke-10 menutup *file.*

Baris ke-11 membuat variabel baru dengan nama *cleaned* berupa array 2 dimensi yang berisi elemen *init* yang memiliki panjang kurang dari 100.

Kode program tersebut diawali dengan deklarasi variabel *games* berupa *list* dan *temp* berupa *string* kosong, dilanjutkan dengan membuka fail “huft.txt” sebagai variabel *file.* Selanjutnya pada baris ke-3 sampai dengan baris ke-10, dilakukan perulangan yang melalui setiap baris dalam *file,* jika baris bukan merupakan *string* kosong, maka baris dibersihkan dari *string* yang tidak diinginkan. Terdapat 3 kesalahan penulisan yang ditemukan pada data yaitu *string* “$2”, “$4”, dan “$6”, *string-string* tersebut bukan merupakan gerakan, sehingga akan dihapus dari data, kemudian baris digabungkan dengan *temp.* Sebaliknya, jika baris merupakan *string* kosong, maka *temp* akan disisipkan ke *games* sebagai *array of string* menggunakan fungsi split(), kemudian *temp* di ubah kembali menjadi *string* kosong. Perulangan ini pada dasarnya melakukan pengambilan rangkaian gerakan dalam fail “huft.txt”, yang mana setiap pertandingan dapat memiliki lebih dari 1 baris dan pertandingan yang berbeda akan dipisahkan oleh “\n”, sehingga kode program tersebut menggabungkan setiap baris ke *temp,* hingga ditemukan baris dengan nilai “\n”, menandakan bahwa pertandingan sebelumnya telah selesai, dan baris berikutnya adalah data dari pertandingan yang berbeda, sehingga *temp* dikembalikan menjadi *string* kosong.

Selanjutnya, pada baris ke-13 sampai dengan baris ke-18, dilakukan deklarasi variabel *gamesCleaned* berupa *list*, kemudian dilakukan perulangan yang melalui setiap baris dalam *games,* jika panjang baris kurang dari 100 gerakan, maka baris tersebut disisipkan ke *gamesCleaned,* perulangan akan terus dilakukan hingga *gamesCleaned* mengandung sebanyak 35.000 baris. Perulangan ini pada dasarnya membuang pertandingan dengan jumlah gerakan yang terlalu banyak, dengan alasan yang telah dijelaskan pada subbab 4.x.

### Implementasi Pra-pemrosesan Data

Implementasi dari pra-pemrosesan dilakukan menggunakan bahasa pemrograman Python, melibatkan pembersihan *string,* pembuatan *dictionary* gerakan, pengkodean gerakan menjadi *integer,* pembuatan N-grams, pemberian *pre-padding*, dan pemisahan fitur dan label.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | remove\_list = ['x', '+', '#', '=Q', '=R', '=N', '=B']  raw = copy.deepcopy(cleaned)  for i, game in enumerate(cleaned):    for j, move in enumerate(game):      for l in remove\_list:        move = move.replace(l, "")      if len(move) > 3 and move[0] != "O":        move = move[0] + move[-2] + move[-1]      cleaned[i][j] = move  found = []  dictionary = {}  flippedDict = {}  for game in cleaned:    for move in game:      if move not in found:        found.append(move)  for i, move in enumerate(found):    dictionary[i+1] = move    flippedDict[move] = i+1  for i, data in enumerate(cleaned):      for j, move in enumerate(data):          cleaned[i][j] = flippedDict[move]  seqRaw = [raw[i][:2\*j+3] for i, line in enumerate(cleaned) for j in range(len(line)//2)]  seq = [line[:2\*j+3] for i, line in enumerate(cleaned) for j in range(len(line)//2)]  for i, game in enumerate(seq):    for j in range(99 - len(game)):      seq[i].insert(0, 0)  label = [sequence.pop() for sequence in seq] |

Kode Program 5.3 Implementasi Pra-pemrosesan Data

Penjesalan dari masing-masing baris pada Kode Program 5.3 adalah sebagai berikut:

Baris ke-1 merupakan deklarasi variabel *remove\_list* berupa *array* 1 dimensi yang berisik *string-string* yang akan dihapus.

Baris ke-2 merupakan deklarasi variabel *raw* yang merupakan duplikat dari *cleaned*

Baris ke-3 melakukan perulangan menggunakan fungsi enumerate() untuk melalui setiap elemen pada *cleaned* beserta indeksnya yang disimpan pada variabel *game* dan *i.*

Baris ke-4 melakukan perulangan menggunakan fungsi enumerate() untuk melaluia setiap elemen pada *game* beserta indeksnya yang disimpan pada variabel *move* dan *j.*

Baris ke-5 melakukan perulangan yang melalui setiap elemen pada *remove\_list* yang disimpan pada variabel *l*

Baris ke-6 menghapus *l* dari *move*

Baris ke-7 melakukan seleksi kondisi jika panjang *move* lebih dari 3 karakter dan tidak dimulai dengan karakter “o”

Baris ke-8 menghapus karakter *move* di antara karakter ke-1 dan karakter ke-2 sebelum terakhir.

Baris ke-9 mengganti elemen *cleaned* pada indeks ke-*ij* dengan *move*

Baris ke-10 merupakan deklarasi variabel *found* berupa *list* untuk menampung gerakan-gerakan pada *cleaned*

Baris ke-11 merupakan deklarasi variabel *dictionary* berupa *dict* sebagai kamus gerakan *integer* ke *string*

Baris ke-12 merupakan deklarasi variabel *flippedDict* berupa *dict* sebagai kamus gerakan *string* ke *integer*

Baris ke-13 melakukan perulangan melalui setiap pertandingan *cleaned* yang disimpan pada variabel *game*

Baris ke-14 melakukan perulangan melalui setiap gerakan *game* yang disimpan pada variabel *move*

Baris ke-15 melakukan seleksi kondisi jika *move* tidak ada dalam *found*

Baris ke-16 menambahkan *move* ke *found* jika seleksi kondisi baris ke-15 tidak terpenuhi

Baris ke-17 melakukan perulangan menggunakan fungsi enumerate() untuk melalui setiap gerakan pada *found* beserta indeksnya yang disimpan pada variabel *move* dan *i*

Baris ke-18 menambahkan elemen baru ke *dictionary* dengan *i*+1 sebagai *key* dan *move* sebagai *value.*

Baris ke-19 menambahkan elemen baru ke *flippedDict* dengan *move* sebagai *key* dan *i*+1 sebagai *value.*

Baris ke-20 melakukan perulangan menggunakan fungsi enumerate() untuk melalui setiap pertandingan pada *cleaned* beserta indeksnya yang disimpan pada variabel *data* dan *i*

Baris ke-21 melakukan perulangan menggunakan fungsi enumerate() untuk melalui setiap gerakan pada *data* beserta indeksnya yang disimpan pada variabel *move* dan *j*

Baris ke-22 mengganti elemen *cleaned* pada indeks ke-*ij* dengan *value* pada *flippedDict* yang memiliki *key move.*

Baris ke-23 dan ke-24 memisahkan setiap pertandingan pada *raw* yang memiliki panjang N menjadi baris, dengan panjang masing-masing baris mengikuti persamaan , kemudian menyimpannya pada variabel *seqRaw*

Baris ke-25 dan ke-26 memisahkan setiap pertandingan pada *cleaned* yang memiliki panjang N menjadi baris, dengan panjang masing-masing baris mengikuti persamaan , kemudian menyimpannya pada variabel *seq*

Baris ke-27 melakukan perulangan menggunakan fungsi enumerate() untuk melalui setiap pertandingan pada *seq* beserta indeksnya yang disimpan pada variabel *game* dan *i*

Baris ke-28 melakukan perulangan melalui setiap nilai *j* dari 0 sampai dengan panjang *game*

Baris ke-29 menyisipkan angka 0 ke *game* pada indeks ke-0

Baris ke-30 memisahkan elemen terakhir dari setiap pertandingan pada *game* dan menyimpannya pada variabel *label*

Kode Program diawali dengan melakukan *import library* copy yang kemudian digunakan untuk menyimpan duplikat dari variabel *gamesCleaned* ke variabel *gamesOri,* yang nantinya akan digunakan saat tahap pengujian model. Pembersihan gerakan dari karakter yang tidak diinginkan dilakukan pada baris ke-3 sampai dengan baris ke-15.

Pembuatan kamus gerakan dilakukan pada baris ke-17 hingga baris ke-23. Diawali dengan deklarasi variabel *found* berupa *list* untuk menyimpan gerakan yang telah ditemukan, variabel *dictionary* berupa *dict* sebagai kamus *integer* ke *string,* dan variabel *flippedDict* berupa *dict* sebagai kamus *string* ke *integer.* Selanjutnya, dilakukan perulangan bersarang untuk melalui elemen-elemen dari *games.* Perulangan pertama melalui setiap baris dalam *games* yang kemudian disimpan pada variabel *game,* dan perulangan kedua melalui setiap elemen dalam *game* yang kemudian disimpan pada variabel *move.* Pada setiap nilai *move,* diperiksa apakah *move* sudah terdapat di dalam *found,* jika tidak ditemukan, maka *move* akan disisipkan ke *found.* Selanjutnya, dilakukan perulangan menggunakan fungsi enumerate()untuk mengambil setiap indeks dan elemen pada *found,* yang kemudian disimpan pada variabel *i* dan *move.* Pada setiap nilai *move,* ditambahkan baris baru pada *dictionary* dengan *i*+1 sebagai *key,* dan *move* sebagai *value.* Ditambahkan pula baris baru ke *flippedDict* dengan *move* sebagai *key* dan *i*+1 sebagai *value.*

Pengkodean gerakan menjadi *integer* dilakukan pada baris ke-24 hingga baris ke-26. Diawali dengan perulangan bersarang untuk melalui elemen-elemen dari *gamesCleaned.* Perulangan pertama menggunakan fungsi enumerate()untuk mengambil indeks dan nilai dari setiap baris, yang kemudian disimpan pada variabel *i* dan *game.* Perulangan kedua juga menggunakan fungsi enumerate()untuk mengambil indeks dan nilai dari setiap elemen *game,* yang kemudian disimpan pada variabel *j* dan *move.* Setiap nilai *move* kemudian ditukar dengan *value* pada *flippedDict* yang memiliki *key* bernilai *move*.

Pembuatan *N-grams* dilakukan pada baris ke-27 sampai dengan baris ke-31. Diawali dengan dua perulangan bersarang untuk melalui elemen-elemen dalam *gamesCleaned*. Perulangan pertama menggunakan fungsi enumerate() untuk mengambil setiap indeks dan nilai baris dari *gamesCleaned,* yang kemudian disimpan pada variabel *i* dan *line*. Perulangan kedua melalui inkremen *j*=0 sampai dengan *j*=(panjang *line*-1)/2. Pada setiap nilai *j,* disisipkan elemen *line* pada indeks ke-0 sampai dengan indeks ke-(2*j*+1) ke *sequences,* kemudian disisipkan elemen *gamesOri* pada indeks ke-[i][0] sampai dengan indeks ke-[i][2*j*+1] ke *sequencesOri.* Kode program ini membuat 2 set *N-grams*, yaitu *sequences* yang berisi *N-grams* dengan gerakan yang telah dibersihkan, dan *sequencesOri* yang berisi *N-grams* dengan gerakan yang belum dibersihkan. *Sequences* akan digunakan pada tahap pelatihan model, sedangkan *sequencesOri* akan digunakan pada tahap pengujian model.

Pemberian *pre-padding* dilakukan pada baris ke-32 hingga baris ke-39. Diawali dengan perulangan bersarang untuk melalui elemen-elemen pada *sequences.* Perulangan pertama menggunakan fungsi enumerate() untuk mengambil setiap indeks dan nilai baris pada *sequences,* yang kemudian disimpan pada variabel *i* dan *game,* perulangan kedua melalui inkremen *j*=0 hingga j=99, sesuai dengan panjang maksimal dari *sequences*. Pada setiap nilai *j,* disisipkan nilai 0 ke *temp* sebanyak (100-panjang baris), kemudian *game* disisipkan ke *temp.* Terakhir, *temp* disisipkan ke *seqPadded* sebagai baris baru. Pada akhir proses ini, akan terbentuk *list seqPadded* dengan panjang setiap baris yang sama, yaitu 99.

Pemisahan fitur dan label dilakukan pada baris ke-41 sampai dengan baris ke-48. Diawali dengan deklarasi variabel *fitur* dan *label* berupa *list,* kemudian dilakukan perulangan yang melalui setiap baris dalam *seqPadded.* Indeks terakhir dalam baris kemudian disisipkan ke *label,* sedangkan sisanya disisipkan ke *fitur.*

### Kelas Model

Implementasi dari kelas model dilakukan dengan mewariskan modul nn.Module milik Torch. Terdapat 2 *method* yang diimplementasikan dari modul tersebut, yaitu method \_\_init\_\_() dan method forward(). Implementasi dari kelas model dapat dilihat pada Kode Program 4.3.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | class RNN(nn.Module):  def \_\_init\_\_(self, input\_size, hidden\_size, num\_layers, num\_classes, embed\_dim):  super(RNN, self).\_\_init\_\_()  self.num\_layers = num\_layers  self.hidden\_size = hidden\_size  self.embed\_dim = embed\_dim  self.embed = nn.Embedding(num\_classes, embed\_dim)  self.lstm = nn.LSTM(embed\_dim, hidden\_size, num\_layers, batch\_first=True)  self.flatten = nn.Flatten()  self.fc1 = nn.Linear(64, 128)  self.fc = nn.Linear(128, num\_classes)  def forward(self, x):  h0 = torch.zeros(self.num\_layers,  x.shape[1],  self.hidden\_size).to(device)  c0 = torch.zeros(self.num\_layers,  x.shape[1],  self.hidden\_size).to(device)  embed = self.embed(x.long())  out, \_ = self.lstm(embed[0].float(), (h0, c0))  out = out[:, -1, :]  out1 = self.flatten(out)  out2 = self.fc1(out1)  out = self.fc(out2)  return out |

Kode Program 5.4 Implementasi Kelas Model

Penjelasan dari masing-masing baris pada Kode Program 5.4 adalah sebagai berikut:

Baris ke-1 merupakan deklarasi kelas RNN yang mewariskan modul nn.Module milih Torch

Baris ke-2 dan ke-3 merupakan deklarasi fungsi konstruktor kelas dengan parameter *self, input\_size, hidden\_size, num\_layers,* dan *num\_classes*

Kode program diawali dengan deklarasi kelas ChessLSTM yang mewarisnya modul nn.Module milik Torch. Selanjutnya, dilakukan implementasi *method* \_\_init\_\_() dengan parameter self, input\_size, hidden\_size, num\_layers, dan num\_classes. Pemisahan fitur dan label dilakukan pada baris ke-41 sampai dengan baris ke-48. Diawali dengan deklarasi variabel *fitur* dan *label* berupa *list,* kemudian dilakukan perulangan yang melalui setiap baris dalam *seqPadded.* Indeks terakhir dalam baris kemudian disisipkan ke *label,* sedangkan sisanya disisipkan ke *fitur.*

### Pelatihan dan Pengujian Model

Tahap pelatihan model melibatkan inisialisasi model dengan memanggil fungsi inisialisasi(), pembelajaran data latih , pembaruan bobot model, dan prediksi data uji. Kode semu untuk tahap pelatihan model dapat dilihat pada Kode Program 4.5.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102 | engine = chess.engine.SimpleEngine.popen\_uci("/content/stockfish/stockfish-ubuntu-x86-64-avx2")  model = RNN(input\_size, hidden\_size, num\_layers, num\_classes).to(device)  criterion = nn.CrossEntropyLoss()  optimizer = torch.optim.Adam(model.parameters())  n\_total\_steps = len(dataloader)  for epoch in range(num\_epochs):  i = 0    for images, labeltot in dataloader:    images = images.reshape(1,batch\_size, input\_size).to(device)      outputs = model(images)      loss = criterion(outputs, labeltot.long())      optimizer.zero\_grad()      loss.backward()      optimizer.step()      i+=1      if (i+1) % 100 == 0:  print (f'Epoch [{epoch+1}/{num\_epochs}], Step [{i+1}/{n\_total\_steps}], Loss: {loss.item():.4f}')      name = str(epoch+1)+".pt"      torch.save(model.state\_dict(), name)      with torch.no\_grad():        sorted = []        criterion2 = nn.CrossEntropyLoss()        res = torch.zeros(len(seq[split:]), 499)        exp = torch.zeros(len(seq[split:]))        n\_correct = 0        acl=0        j = 0        illegal = 0  for rangkaian, next in zip(torch.tensor(seq[split:]), torch.tensor(label[split:])):  rangkaian = rangkaian.reshape(1, 1, input\_size).to(device)          logitsRangkaian = model(rangkaian)          \_, prediksi = torch.max(logitsRangkaian.data, 1)          res[j] = logitsRangkaian          exp[j] = next          sorted.append(np.argsort(logitsRangkaian[0].numpy())[::-1])          max = float('-inf')          board = chess.Board()          for kintil in seqRaw[split+j][:-1]:            board.push\_san(kintil)          info = engine.analyse(board, chess.engine.Limit(depth=5))          temp = info["score"].white().score(mate\_score=1500)          move = dictionary[prediksi.item()]          try: board.parse\_san(move)          except chess.AmbiguousMoveError:            squares = board.pieces(is\_ambiguous(move)[1], 1)            for square in squares:              try:  board.find\_move(square, chess.parse\_square(move[-2:]))               move = (move[0] + chess.square\_name(square) + move[-2:])              except: pass          except chess.IllegalMoveError:            if is\_illegal(move):              illegal += 1            for index in sorted[j]:              if index == 0:                continue              if is\_illegal(dictionary[index]):                continue              if is\_ambiguous(dictionary[index]):           squares = board.pieces(is\_ambiguous(dictionary[index])[1], 1)                for square in squares:                  try:  board.find\_move(square, chess.parse\_square(dictionary[index][-2:]))  move = (dictionary[index][0] + chess.square\_name(square) + dictionary[index][-2:])                  except: pass                break              if is\_promotion(dictionary[index]):                moveTemp = dictionary[index] + "=Q"                if is\_illegalAbs(temp):                  continue                move = moveTemp                break              else:                move = dictionary[index]                break          board.push\_san(move)         info2 = engine.analyse(board, chess.engine.Limit(depth=5))          temp2 = info2["score"].white().score(mate\_score=1500)          acl += (temp - temp2)          j+=1          if j%10000 == 0:            print(f'{j}, {temp - temp2}, {acl/j}')          if move == dictionary[next.item()]:            n\_correct += 1        losss = criterion2(res, exp.long())        acc = 100.0 \* n\_correct / j      print(f'loss {losss}, acc: {acc}, acl: {acl/j}, illegal: {illegal}/{j}') |

Kode Program 5.5 Implementasi Pelatihan dan Pengujian Model

Penjelasan dari masing-masing baris pada Kode Program 5.5 adalah sebagai berikut:

Baris ke-1 sampai dengan ke-3 memuat Stockfish ke variabel *engine* untuk yang nantinya akan digunakan untuk menghitung ACPL data uji

Baris ke-4 melakukan inisialisasi objek model yang disimpan pada variabel *model*

Baris ke-6 melakukan inisialisasi objek *CrossEntropyLoss* yang disimpan pada variabel CRITERIOOOONNNNNNNNN

Baris ke-7 merupakan inisialisasi objek *optimizer* adam yang disimpan pada variabel *optimizer*

Skip

Baris ke-9 melakukan perulangan sebanyak nilai *num\_epoch*

Baris ke-10 melakukan deklarasi variabel *i* untuk menyimpan indeks dari baris data latih yang sedang diproses, diinisialisasikan sebagai 0

Baris ke-11 melakukan perulangan melalui setiap elemen dalam *dataloader* beserta dengan indeksnya yang disimpan pada variabel *IMAGE* dan *LABELTOOOOOOOTTTTTTTTTTTT*

Baris ke-12 mengubah dimensi **IMAGES** menjadi (1,*batch\_size*,*input\_size*)

Baris ke-13 memprediksi **IMAGES** menggunakan *model* dan menyimpan hasil prediksi pada variabel *outputs*

Baris ke-14 menghitung *loss* dari *outputs* dan **LABELTOT**

**SKIP**

**SKIP**

**SKIP**

**SKIP**

### Mesin Catur

Implementasi dari mesin catur dapat dilihat pada Kode Program 4.4.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137 | hidden\_size = 64  num\_classes = 499  embed\_dim = 32  input\_size = 98  num\_layers = 1  class myLSTM(nn.Module):  def \_\_init\_\_(self, input\_size, hidden\_size, num\_layers, num\_classes):  super(myLSTM, self).\_\_init\_\_()  self.num\_layers = num\_layers  self.hidden\_size = hidden\_size  self.embed\_dim = embed\_dim  self.embed = nn.Embedding(num\_classes, embed\_dim)  self.lstm = nn.LSTM(embed\_dim, hidden\_size, num\_layers, batch\_first=True)  self.fc = nn.Linear(hidden\_size, num\_classes)  def forward(self, x):  h0 = torch.zeros(self.num\_layers,x.shape[1],self.hidden\_size)  c0 = torch.zeros(self.num\_layers,x.shape[1],self.hidden\_size)  embed = self.embed(x.long())  out, \_ = self.lstm(embed[0].float(), (h0, c0))  out2 = out[:, -1, :]  out5 = self.fc(out2)  return out5  model2 = myLSTM(input\_size, hidden\_size, num\_layers, num\_classes)  model2.load\_state\_dict(torch.load('ingite.pt'))  model2.eval()  def encode(games):  print(f'input: {games}')  game = games  for j, move in enumerate(game):  temp = move.replace("x", "")  temp = temp.replace("+", "")  temp = temp.replace("#", "")  temp = temp.replace("0!", "")  temp = temp.replace("!", "")  temp = temp.replace("00", "")  temp = temp.replace("=R", "")  temp = temp.replace("=Q", "")  temp = temp.replace("=N", "")  temp = temp.replace("=B", "")  if (len(temp) > 3) and (temp[0] != "O"):  temp = temp[0] + temp[-2] + temp[-1]  try:  game[j] = flippedDict[temp]  except KeyError:  game[j] = 0  print(game)  for j in range(98 - len(game)):  game.insert(0, 0)  return game  def is\_promotion(move):  if move.islower() and (move[-1] == "1" or move[-1] == "8"):  return True  else:  return False  def is\_ambiguous(move):  piece = {"N": 2, "B": 3, "R": 4, "Q": 5}  try:  board.parse\_san(move)  except chess.AmbiguousMoveError:  return (True, piece[move[0]])  except chess.IllegalMoveError:  return False  else:  return False  def is\_illegal(move):  try:  board.parse\_san(move)  except chess.IllegalMoveError:  if is\_promotion(move):  return False  else:  return True  except chess.AmbiguousMoveError:  return False  def is\_illegalAbs(move):  try:  board.parse\_san(move)  # print(f"{move} is illegal")  except:  return True  board = chess.Board()  while True:  msg = input()  if msg == "uci":  print("uciok")  elif msg == "isready":  print("readyok")  elif msg==("ucinewgame"):  continue  elif msg.startswith("position startpos"):  board.reset()  if len(msg.split()) <=3:  choice = ["e4", "d4"]  move = choice[random.randint(0,1)]  continue  moves = msg.split(" ")[3:]  moves2 = []  for move in moves:  moves2.append(board.san(board.parse\_san(move)))  board.push\_san(move)  # print(f'woi {moves2}')  data = encode(moves2)  data = torch.tensor([[data]])  print(data)  nextMove = model2(data)  sorted = np.argsort(nextMove[0].detach().numpy())[::-1]  prediksi = np.argmax(nextMove.detach())  move = dictionary[prediksi.item()]  print(move)  max = float("-inf")  try:  board.parse\_san(move)  except chess.AmbiguousMoveError:  squares = board.pieces(is\_ambiguous(move)[1], 1)  for square in squares:  try:  board.find\_move(square, chess.parse\_square(move[-2:]))  move = (move[0] + chess.square\_name(square) + move[-2:])  except:  pass  except chess.IllegalMoveError:  # move is undefined promotion  for index in sorted:  if index == 0:  continue  if is\_illegal(dictionary[index]):  continue  if is\_ambiguous(dictionary[index]):  squares = board.pieces(is\_ambiguous(dictionary[index])[1], 1)  for square in squares:  try:  board.find\_move(square, chess.parse\_square(dictionary[index][-2:]))  move = (dictionary[index][0] + chess.square\_name(square) + dictionary[index][-2:])  except:  pass  break  if is\_promotion(dictionary[index]):  moveTemp = dictionary[index] + "=Q"  if is\_illegalAbs(moveTemp):  continue  move = moveTemp  break  else:  move = dictionary[index]  break  elif msg.startswith("go"):  print(f"bestmove {board.push\_san(move)}")  elif msg == "quit":  break |

Kode Program 5.6 Implementasi Mesin Catur

Penjelasan dari masing-masing baris pada Kode Program 5.5 adalah sebagai berikut:

Baris ke-1 sampai dengan ke-94 merupakan deklarasi kembali variabel dan fungsi yang terdapat pada Kode Program X.X karena mesin catur dibangun pada lingkungan pemrograman yang berbeda dan akan dikonversikan menjadi fail *executable*

# hasil dan analisis

## Hasil *Hyperparameter Tuning*

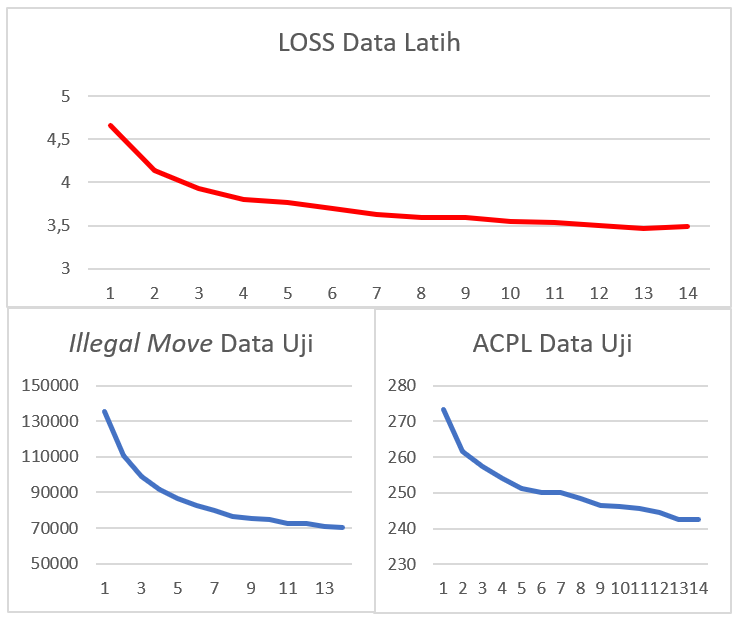
Berdasarkan 11 skenario *hyperparameter tuning* yang telah dilakukan*,* didapatkan nilai ACPL masing-masing skenario sebagaimana dapat dilihat pada Tabel 6.1. Berdasarkan hasil tersebut didapatkan nilai *hyperparameter* yang menghasilkan nilai ACPL terendah yaitu nilai *batch size* sebesar 500, nilai *hidden size* sebesar 64, nilai *embedding dimension* sebesar 32, dan nilai *num layers* sebesar 1. Nilai-nilai *hyperparameter* tersebut kemudian akan digabungkan untuk membentuk arsitektur akhir model LSTM yang akan digunakan untuk tahap pelatihan model.

Tabel 6.1 Hasil *Hyperparameter Tuning*

|  |  |  |  |
| --- | --- | --- | --- |
| **Skenario** | **No. skenario** | **Nilai yang diuji** | **Nilai ACPL** |
| *Batch size* | 1. | 250 | 225 |
| **2.** | **500** | **212** |
| 3. | 1000 | 230 |
| *Hidden size* | 4. | 32 | 235 |
| **5.** | **64** | **212** |
| 6. | 128 | 215 |
| *Embedding dimension* | **7.** | **32** | **198** |
| 8. | 64 | 212 |
| 9. | 128 | 223 |
| *Num layers* | **10.** | **1** | **212** |
| 11. | 2 | 231 |

## Hasil Pelatihan Model

Pelatihan model dilakukan sebanyak 14 *epoch* sebelum dihentikan oleh *callback* karena nilai ACPL data uji tidak lagi mengalami penurunan*.* Selama 14 *epoch* pelatihan, *loss* data latih mengalami penurunan dari angka 4,66 hingga 3,47, nilai ACPL data uji mengalami penurunan dari angka 273,49 hingga 242,57, sedangkan jumlah *illegal move* dari prediksi data uji mengalami penurunan dari angka 135.237 hingga 70.598, sebagaimana dapat dilihat pada grafik-grafik di Gambar 6.1. Rincian dari setiap *epoch* pelatihan dapat dilihat pada Tabel 6.1. Secara keseluruhan, pelatihan model menghasilkan nilai *loss* dan nilai ACPL yang masih cukup tinggi. Kegagalan pelatihan model dalam meminimalkan nilai *loss* dan ACPL ini akan dibahas lebih lanjut pada analisis hasil di Subbab 6.4.



Gambar 6.1 Grafik Pelatihan dan Pengujian Model

Tabel 6.2 Hasil *Hyperparameter Tuning*

|  |  |  |  |
| --- | --- | --- | --- |
| ***Epoch* ke** | ***Loss* Data Latih** | **ACPL data uji** | ***illegal move* data uji** |
| 1 | 4,6582 | 273,4874 | 135.237 |
| 2 | 4,1387 | 261,6148 | 110.596 |
| 3 | 3,9346 | 257,429 | 99.100 |
| 4 | 3,8002 | 253,9827 | 91.609 |
| 5 | 3,7615 | 251,2748 | 86.747 |
| 6 | 3,6994 | 250,2338 | 82.469 |
| 7 | 3,6264 | 250,2214 | 79.801 |
| 8 | 3,5954 | 248,3863 | 76.506 |
| 9 | 3,5919 | 246,4166 | 75.572 |
| 10 | 3,5426 | 246,1697 | 74.889 |
| 11 | 3,5326 | 245,6245 | 72.510 |
| 12 | 3,5026 | 244,4351 | 72.375 |
| 13 | 3,4672 | 242,5739 | 71.138 |
| 14 | 3,4884 | 242,6341 | 70.598 |

## Hasil Pengujian Melawan Mesin Catur Lain

Pengujian melawan mesin catur lain dilakukan dengan melawan mesin catur Komodo Level 1, Maia Chess 1, dan Stockfish 16, dengan jumlah pertandingan sebanyak 50 pertandingan untuk masing-masing lawan. Pemeriksaan lebih lanjut terhadap pertandingan-pertandingan tersebut menunjukkan bahwa dari 50 pertandingan yang dilakukan, terdapat beberapa pertandingan yang merupakan duplikat dari pertandingan lainnya. Pertandingan duplikat ini kemudian dihilangkan dari data dengan bantuan perangkat lunak PGN-Extract. Selanjutnya, data pertandingan yang telah dibersihkan dimuat ke fitur *import game* milik Lichess.com untuk melihat akurasi yang didapatkan oleh mesin catur pada masing-masing pertandingan. Rincian dari hasil pengujian melawan mesin catur lain dapat dilihat pada Tabel 6.X.

Tabel 6.3 Hasil Pengujian Melawan Mesin Catur Lain

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Nama Lawan** | **Estimasi Rating Lawan** | **Skor**  **(menang-kalah-seri)** | **Akurasi Tertinggi** | **Akurasi Terendah** | **Rata-Aata Akurasi** |
| Komodo level 1 | ~100 | 0-18-6 | 61% | 23% | 45,75% |
| Maia Chess 1 | ~1100 | 0-23-1 | 80% | 52% | 65,54% |
| Stockfish 16 | ~3000 | 0-43-0 | 90% | 61% | 78,98% |

## Analisis Hasil

Pengujian melawan mesin catur lain menunjukkan kinerja mesin catur yang jauh dari kata memuaskan. Namun, analisis lebih lanjut terhadap pertandingan-pertandingan pengujian menunjukkan bahwa mesin catur sebenarnya bermain dengan cukup baik pada awal pertandingan, dengan melakukan gerakan-gerakan yang bermakna dan masuk akal. Namun, memasuki pertengahan pertandingan atau fase *middle game*, mesin catur seolah mulai kehilangan konteks dan situasi dari papan catur, sehingga mesin catur mulai melakukan gerakan-gerakan yang tidak akurat. Pada fase *middle game* ini, mesin catur kerap melakukankesalahan-kesalahan seperti membangun serangan pada petak yang dijaga oleh lawan atau melakukan *trade* yang tidak menguntungkan. Secara keseluruhan, gerakan-gerakan yang dilakukan mesin catur pada fase *middle game* ini masih memiliki maksud dan tujuan yang dapat dipahami, namun mesin catur gagal dalam memperhitungkan respons apa yang dapat dilakukan oleh lawan, serta dampak yang dapat ditimbulkan oleh gerakan tersebut. Terlihat saat melawan Komodo Level 1, mesin catur beberapa kali mampu membangun keunggulan pada awal pertandingan, namun tidak dapat mempertahankan keunggulan tersebut pada fase *middle game*. Mendekati akhir pertandingan atau fase *endgame*, mesin catur telah kehilangan sepenuhnya konteks dari papan, dan hanya menggerakkan buah raja, melakukan gerakan yang sama berulang kali, atau mendorong pion tanpa tujuan yang jelas.

Hilangnya pemahaman dan kesadaran mesin catur terhadap konteks papan tersebut dapat disebabkan oleh data yang tidak konsisten. Data pada penelitian ini diambil dari pertandingan-pertandingan yang dimainkan oleh lebih dari 16.000 pemain di Lichess.com. Masing-masing pemain tersebut akan memiliki tingkat kemampuan, preferensi buah catur, strategi bermain, dan cara berpikir yang berbeda-beda pula. 2 orang pemain dapat memainkan posisi yang sama persis dengan cara yang sangat berbeda, menyebabkan data yang tidak konsisten untuk dipelajari oleh model LSTM.

Hal tersebut tidak begitu terlihat pada fase awal pertandingan, karena pada awal sebuah pertandingan, pemain catur tidak begitu bergantung kepada kemampuan analisis posisi papan atau penyusunan taktik dan serangan, melainkan lebih bergantung kepada **mengingat** dan mengikuti pola gerakan yang telah dirumuskan sebelumnya, yang disebut juga sebagai *opening book*. Sebagai contoh e4 e5 Nf3 Nc6 d4 disebut dengan *scotch opening,* sedangkan e4 d5 exd5 Qxd5 Nc3 disebut dengan *Scandinavian opening*. Ketergantungan terhadap *opening book* serta sedikitnya jumlah gerakan bermakna yang dapat dilakukan di awal pertandingan membuat model LSTM masih dapat menangkap dan mengenali pola gerakan dengan cukup baik di awal pertandingan. Memasuki fase pertengahan pertandingan atau *middle game,* pemain mulai mengandalkan kemampuan analisis papan dan penyusunan taktik mereka untuk melakukan gerakan, dan tidak lagi bergantung kepada *opening book*. Hal ini membuat data menjadi subjektif terhadap cara berpikir masing-masing pemain, sehingga data menjadi tidak konsisten saat pertandingan mulai memasuki fase *middle game*.

Data yang tidak konsisten tidak dapat dipelajari dengan baik oleh model LSTM, hal ini terlihat pula pada nilai *loss* yang masih tinggi pada saat tahap pelatihan model. Selain itu, data yang diambil dari pertandingan yang dilakukan oleh manusia juga menyebabkan mesin catur tidak bermain dengan baik saat lawan melakukan gerakan yang tidak terduga atau tidak umum dilakukan oleh manusia. Sebagai contoh, jika hitam memulai pertandingan dengan gerakan yang jarang dimainkan oleh manusia seperti h5 atau a5, mesin catur cenderung untuk memberikan respons yang tidak akurat, atau bahkan sepenuhnya mengabaikan gerakan tersebut. Data pada penelitian ini juga tidak mempertimbangkan kemungkinan bahwa putih dapat berada pada posisi yang tidak unggul selama 90% dari pertandingan dan hanya unggul pada 10% di fase *endgame*, kondisi tersebut akan menyebabkan model LSTM mempelajari pertandingan yang mengandung 90% gerakan tidak akurat oleh putih. Berdasarkan hal-hal tersebut, penggunaan data yang lebih konsisten dan lebih berkualitas, serta pemrosesan data yang lebih teliti dapat dilakukan untuk mendapatkan hasil yang lebih baik.

Saat dilakukan uji coba lebih lanjut, terlihat bahwa saat diberikan posisi *mate-in-one*, yaitu posisi di mana mesin catur dapat melakukan sekakmat pada gerakan berikutnya, mesin catur beberapa kali dapat menemukan gerakan sekakmat tersebut. Pola *mate-in-one* yang dapat diidentifikasi oleh mesin catur ini adalah pola-pola sekakmat sederhana seperti beberapa variasi dari *scholar's mate* dan *fool's mate*. Meskipun begitu, pada saat bertanding mesin catur tidak memiliki kemampuan untuk membangun serangan yang padu untuk mencapai posisi *mate-in-one* tersebut.

Hal menarik lain yang terlihat dari hasil pengujian adalah angka akurasi pengujian yang menunjukkan bahwa mesin catur seolah bermain dengan akurasi yang lebih tinggi saat bermain melawan mesin catur yang lebih kuat. Terlihat saat melawan mesin catur Komodo level 1 yang memiliki estimasi *rating* sebesar 100, mesin catur mendapatkan rata-rata akurasi sebesar 40,x dari 24 pertandingan. Sementara itu, saat melawan mesin catur Stockfish yang memiliki estimasi *rating* sebesar 3000, mesin catur mendapatkan rata-rata akurasi sebesar 75,x% dari 20 pertandingan. Peningkatan akurasi ini disebabkan oleh musuh yang lebih kuat hanya membutuhkan mesin catur untuk melakukan beberapa gerakan tidak akurat untuk dapat menang. Terlihat pada beberapa pertandingan pengujian, Stockfish 16 hanya membutuhkan 3 gerakan tidak akurat oleh mesin catur untuk dapat mengalahkannya, sedangkan Maia Chess 1 membutuhkan sekitar 6 sampai 7 gerakan tidak akurat oleh mesin catur untuk dapat mengalahkannya. Hal inilah yang menyebabkan akurasi mesin catur seolah mengalami peningkatan saat bertandingan dengan lawan yang lebih kuat.

# penutup

## Kesimpulan

Berdasarkan 11 skenario *hyperparameter tuning* yang telah dilakukan*,* didapatkan nilai ACPL masing-masing skenario sebagaimana dapat dilihat pada Tabel 6.1. Berdasarkan hasil tersebut didapatkan nilai *hyperparameter* yang menghasilkan nilai ACPL terendah yaitu nilai *batch size* sebesar 500, nilai *hidden size* sebesar 64, nilai *embedding dimension* sebesar 32, dan nilai *num layers* sebesar 1. Nilai-nilai *hyperparameter* tersebut kemudian akan digabungkan untuk membentuk arsitektur akhir model LSTM yang akan digunakan untuk tahap pelatihan model.

DAFTAR REFERENSI

*About Stockfish.* [online] Tersedia di: <https://stockfishchess.org/about/> [Diakses 4 September 2023]

Basheer, I.A. and Hajmeer, M., 2000. Artificial neural networks: fundamentals, computing, design, and application. *Journal of microbiological methods*, *43*(1), pp.3-31.

Hochreiter, S. and Schmidhuber, J., 1997. Long short-term memory. *Neural computation*, *9*(8), pp.1735-1780.

International Chess Federation [FIDE], 2022. *FIDE Laws of Chess taking effect from 1 January 2023*. [online] Tersedia di: <https://handbook.fide.com/chapter/E012023> [Diakses 8 Juli 2023]

Jain, A.K., Mao, J. and Mohiuddin, K.M., 1996. Artificial neural networks: A tutorial. *Computer*, *29*(3), pp.31-44.

Kahlen., 2004. *Description of the universal chess interface (UCI)*. [online] Tersedia di <https://www.wbec-ridderkerk.nl/html/UCIProtocol.html> [Diakses 18 Agustus 2023]

Klein, D., 2022. Neural Networks for Chess. *arXiv preprint arXiv:2209.01506*.

*Lichess Accuracy Metric.* [online] Tersedia di: https://lichess.org/page/accuracy [Diakses 4 September 2023]

Mehta, F., Raipure, H., Shirsat, S., Bhatnagar, S. dan Bhovi, B., 2020. Predicting chess moves with multilayer perceptron and limited Lookahead. *J Eng Res Appl*, *10*(4), pp.05-08.

Panchal, H., Mishra, S. dan Shrivastava, V., 2021, October. Chess moves prediction using deep learning neural networks. In *2021 International Conference on Advances in Computing and Communications (ICACC)* (pp. 1-6). IEEE.

Santosa, R.D.W., Bijaksana, M.A. dan Romadhony, A., 2021. Implementasi algoritma long short-term memory (lstm) untuk mendeteksi penggunaan kalimat abusive pada teks bahasa indonesia. *eProceedings of Engineering*, *8*(1).

Selle, N., Yudistira, N. dan Dewi, C., 2022. Perbandingan Prediksi Penggunaan Listrik dengan Menggunakan Metode Long Short Term Memory (LSTM) dan Recurrent Neural Network (RNN). *Jurnal Teknologi Informasi dan Ilmu Komputer*, *9*(1), pp.155-162.

Shannon, C.E., 1950. XXII. Programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, *41*(314), pp.256-275.

*Standard: Portable Game Notation Specification and Implementation Guide*, 2004. [online] Tersedia di <ia902908.us.archive.org/26/items/pgn-standard-1994-03-12/PGN\_standard\_1994-03-12.txt> [Diakses 28 Agustus 2023]

Yu, Y., Si, X., Hu, C. and Zhang, J., 2019. A review of recurrent neural networks: LSTM cells and network architectures. *Neural computation*, *31*(7), pp.1235-1270.

Linear Transformation:

Y = h1Wt + b

print(torch.randn(6,2))

tensor([[-0.8139, -0.2747],

[-0.4778, -1.4873],

[ 1.4972, 0.9970],

[-2.9437, -1.8984],

[ 0.6662, -0.9028],

[-1.3079, 0.6734]])