BAB II
TINJAUAN PUSTAKA

2.1 Metode Tinjauan Pustaka

Metode tinjauan pustaka adalah langkah penting dalam proses penelitian ilmiah yang melibatkan analisis mendalam terhadap literatur terkait. Pendekatan ini dilakukan dengan cara yang terstruktur untuk mengumpulkan, mengevaluasi, dan menganalisis literatur yang relevan. Tujuannya adalah untuk memahami topik yang sedang diteliti dengan lebih komprehensif. Tinjauan pustaka bukan hanya sekadar membaca literatur, tetapi lebih kepada usaha kritis dalam mengevaluasi temuan-temuan sebelumnya yang relevan [19].

Penggunaan metode tinjauan pustaka dalam konteks penelitian memberikan panduan yang kuat untuk memahami perkembangan ilmu dan penelitian dalam suatu disiplin. Menurut Kumar, tinjauan pustaka adalah proses pemeriksaan dan evaluasi yang cermat terhadap literatur yang relevan untuk menemukan, menilai, dan menginterpretasi temuan penelitian sebelumnya yang relevan dengan topik penelitian tertentu [20]. Pendekatan ini melibatkan pengumpulan literatur dari berbagai sumber yang tepercaya, seperti *paper* dari jurnal ilmiah, *paper* dari *conference* (*proceeding*), tesis, disertasi, *report* (laporan) dari organisasi yang terpercaya, dan buku untuk mendapatkan wawasan yang komprehensif [21].

Dalam proses penulisan laporan tugas akhir ini, metode tinjauan pustaka yang digunakan adalah *Systematic Literature Review* (*SLR*). *SLR*, atau yang dikenal sebagai Tinjauan Pustaka Sistematis, adalah suatu pendekatan metodologi yang memiliki peran penting dalam menghimpun, menilai, dan menggabungkan literatur ilmiah terkait suatu topik secara sistematis dan terstruktur [20]. Melalui pendekatan ini, peneliti dapat mengidentifikasi dan menganalisis temuan-temuan yang relevan dari berbagai sumber literatur dengan cara yang komprehensif.

2.2 Studi Literatur Penelitian Terdahulu

Tabel 2.1 Studi Literatur Penelitian Terdahulu

Penulis	Judul	Masalah	*Dataset*	Metode	Hasil
J. Saxe dan K. Berlin [15] (2016)	*Deep Neural Network Based Malware Detection Using Two Dimensional Binary Program Features*	Perlu dilakukan eksplorasi lebih lanjut terhadap fitur-fitur yang digunakan, optimasi *hyperparameter*, serta penggunaan model yang lebih kompleks.	VirusTotal	*Deep Neural Network* yang dilatih dengan algoritma *back propagation*	Akurasi = 95% dan *false positive rate* = 0,1%
Huda et al. [22] (2016)	*Android malware detection using back-propagation neural network*	Penggunaan sampel data yang terbatas dan kegunaan model dalam kondisi yang lebih luas.	*android manifest*	*Multilayer Perceptron Neural Network* yang dilatih dengan algoritma *back propagation*	Metode yang diusulkan efektif dalam mengenali sampel *malware* yang ada.
Pan, et al. [23] (2016)	*Malware classification based on the behavior analysis and back propagation neural network*	Perlu dilakukan eksplorasi lebih lanjut dalam hal jumlah sampel data, optimasi *hyperparameter*, serta penggunaan model yang lebih kompleks.	Forum Kafan	*Back Propagation Neural Network (BPNN)*	Akurasi = 86%.
Makandar dan Patrot [16] (2016)	*Malware analysis and classification using Artificial Neural Network*	Kesulitan mengelola vektor fitur berdimensi tinggi secara efektif sehingga berdampak pada akurasi suboptimal.	Mahenhur *Dataset*	*Artificial Neural Network (ANN)* dan algoritma *back propagation* serta ekstraksi fitur	Akurasi = 96,35%.

				menggunakan *Gabor Wavelet Transform* (*GWT*) dan *Generic Fourier Descriptor* (*GIST*)	
Babak et al. [17] (2018)	*Malware classification and detection using artificial neural network*	Memerlukan pengetahuan yang mendalam tentang karakteristik *malware* untuk melakukan *labeling dataset* dengan benar. Kesalahan dalam *labeling dataset* dapat mengakibatkan *training* model yang tidak akurat.	https://github.com/naisofly/Static-*Malware*-Analysis	*Neural Network* yang dilatih dengan algoritma *back propagation* dan *Gradient Descent*	Akurasi = 97,8%, presisi = 97,6%, dan *recall* = 96,6%.
Ahmed et al. [18] (2022)	*Deep Learning-Based Classification Model for Botnet Attack Detection*	Kompleksitas model yang berpotensi mengalami *overfitting* pada data *training* dan model yang sulit diinterpretasi.	CTU-13	*Deep Neural Network* dan algoritma *back propagation*	Akurasi = 99,6%.

Tabel 2.1 memuat penelitian sebelumnya dengan penerapan model *neural network* dan algoritma *backpropagation* untuk deteksi dan klasifikasi *malware.* Saxe dan Berlin [15] mengemukakan suatu konsep sistem klasifikasi *malware* yang memanfaatkan *neural network* yang memiliki dua lapisan tersembunyi. Lapisan-lapisan tersembunyi tersebut terdiri dari 1024 *Parametric Rectified Linear Units* (*PReLu*), sementara lapisan *output* menggunakan neuron *sigmoid* untuk mengklasifikasikan *malware* atau *benign*. Dalam percobaan ini, *neural network*

dilatih menggunakan metode *backpropagation*. Eksperimen ini berhasil mencapai tingkat deteksi sebesar 95%, dengan *false positive rate* sebesar 0,1%, saat diuji pada *dataset* berisi 400.000 sampel. Meskipun demikian, terdapat potensi untuk meningkatkan kinerja metode ini melalui eksplorasi lebih lanjut terhadap fitur-fitur yang digunakan, optimasi *hyperparameter*, serta penggunaan model yang lebih kompleks.

Huda et al. [22] menggunakan *multilayer perceptron artificial neural network* yang dilatih dengan algoritma *backpropagation* untuk mengklasifikasikan aplikasi Android ke dalam kategori *malware* atau *benign*. Hasil eksperimen menunjukkan bahwa metode yang diusulkan efektif, dengan akurasi yang cukup tinggi dalam mengenali sampel *malware* yang ada. Kelebihan utama dari metode ini adalah mampu mengenali pola-pola yang kompleks pada data. Meskipun demikian, terdapat keterbatasan dalam hal jumlah sampel data yang digunakan dalam penelitian ini dan kemampuan model untuk diterapkan dalam kondisi yang lebih luas.

Pan, et al. [23] mengusulkan suatu pendekatan klasifikasi *malware* yang berbasis pada analisis dinamis. Dalam pendekatan ini, analisis dinamis digunakan untuk merumuskan profil perilaku yang menjadi dasar dalam mengidentifikasi fitur-fitur khas dari *malware*. Eksperimen ini menerapkan model *Back Propagation Neural Network (BPNN)*. Dari pengujian yang dilakukan, pendekatan ini berhasil mencapai tingkat akurasi rata-rata 86%, dengan tingkat akurasi mencapai 99% pada kategori non-*malware*. Meskipun demikian, terdapat ruang untuk meningkatkan kinerja metode ini melalui berbagai cara. Di antaranya adalah dengan menambah jumlah sampel data yang digunakan untuk *training*, melakukan optimasi terhadap *hyperparameter* yang digunakan dalam model, serta mempertimbangkan penggunaan model yang lebih kompleks guna mengatasi variasi yang lebih rumit dari perilaku *malware*.

Makandar dan Patrot [16] melakukan eksplorasi dalam klasifikasi sampel *malware* dengan mengubahnya menjadi gambar *grayscale*, kemudian menerapkan analisis berbasis fitur tekstur. Dalam upaya ini, mereka melakukan ekstraksi sebanyak 512 vektor fitur menggunakan *Gabor Wavelet Transform* (*GWT*) dan

Generic Fourier Descriptor (*GIST*) guna memahami perilaku dari sampel-sampel *malware* tersebut. Dari seluruh vektor fitur tersebut, mereka memilih 320 fitur dua dimensi untuk selanjutnya diterapkan pada metode *Artificial Neural Network (ANN)* dalam proses klasifikasinya. Pemilihan penggunaan *feed-forward artificial neural network* didorong oleh kemampuannya untuk mengidentifikasi pola tersembunyi dalam data kompleks, seperti gambar, suara, dan sinyal, serta memberikan fleksibilitas implementasi. Selama proses *training*, digunakan algoritma *backpropagation*. Meskipun telah berusaha keras, para penulis menghadapi tantangan dalam mengelola vektor fitur berdimensi tinggi secara efektif, yang berdampak pada akurasi suboptimal sebesar 96,35% untuk tugas klasifikasi *malware*.

Babak et al. [17] mengembangkan model *neural network* untuk mengklasifikasikan *file PE,* menggunakan algoritma *Gradient Descent* dengan *decayed learning rate* untuk memperbarui bobot selama proses *training*. Sebelum proses *training* dan *testing* model *neural network* dilakukan, *dataset* yang digunakan di-*labeling* terlebih dahulu dengan langkah-langkah tertentu, termasuk pengelompokkan sampel, pemeriksaan validitas *PE,* pengecualian *malware* terbungkus, dan pemeriksaan melalui VirusTotal API. Proses ini dilakukan untuk memastikan bahwa setiap sampel dalam *dataset* mendapatkan label yang benar. Penelitian ini menerapkan metode *backpropagation*, di mana setiap *forward pass* diikuti oleh *backpropagation*. Model *neural network* yang dibangun menggunakan fungsi aktivasi *softmax* untuk neuron *output*. Dalam eksperimen ini, digunakan *dataset* yang terdiri dari 4.000 sampel (3.000 *file PE malware* dan 1.000 *benign*). Hasil implementasi model *neural network* ini mencapai metrik kinerja yang mengesankan. Dengan rata-rata akurasi sebesar 97,8%, serta presisi dan *recall* berturut-turut mencapai 97,6% dan 96,6%, hasil temuan ini menegaskan kehandalan pendekatan yang diterapkan dalam melakukan klasifikasi yang akurat terhadap *file PE*, memisahkan dengan jelas antara kategori *malware* dan *benign*. Penggunaan algoritma *Gradient Descent* serta penerapan fungsi aktivasi *softmax* juga menunjukkan bahwa model ini memiliki kemampuan dalam menangani data

yang kompleks serta mampu secara efektif mengidentifikasi perbedaan antara *file-file malware* dan *benign*.

Ahmed et al. [18] menggunakan dua model *neural network* yaitu *feed-forward backpropagation ANN*, dimana dalam penelitian ini, algoritma *backpropagation* digunakan dalam pengembangan model tersebut. Model ini juga melibatkan model *deep neural network* dengan optimasi algoritma *Adam*. Penelitian ini menggunakan *dataset* CTU-13 yang mencakup berbagai skenario serangan *botnet* dan lalu lintas normal. Hasil penelitian menunjukkan bahwa model *deep learning* yang diusulkan dapat mengenali serangan *botnet* dengan akurasi tinggi, mencapai 99,6%. Namun, kelemahan metode ini adalah kompleksitas model yang berpotensi mengalami *overfitting* pada data *training* dan model yang sulit diinterpretasi.

Penelitian sebelumnya telah menunjukkan bahwa *Back Propagation Neural Network* memiliki potensi yang menjanjikan dalam mendeteksi *malware*, karena mampu mengenali pola dan hubungan yang rumit dalam data. Akan tetapi, kinerja *Back Propagation Neural Network* sangat dipengaruhi oleh *hyperparameter* yang mengatur struktur jaringan dan proses pembelajaran. Meskipun penelitian sebelumnya telah mengkaji penggunaan *Back Propagation Neural Network* untuk mengklasifikasikan *malware*, penerapan khusus *Back Propagation Neural Network* dengan optimasi *hyperparameter* menggunakan metode *Grid Search* serta penggunaan regularisasi *dropout* masih relatif belum banyak diteliti. Oleh karena itu, penelitian ini mengadopsi dua pendekatan berbeda dalam menerapkan model *neural network* untuk mendeteksi *malware*.

Pendekatan pertama melibatkan penerapan *Back Propagation Neural Network* tanpa optimasi *hyperparameter tuning*. Dalam upaya meningkatkan performa model *Back Propagation Neural Network*, penelitian ini mengusulkan pendekatan kedua yang melibatkan penerapan *Back Propagation Neural Network* dengan optimasi *hyperparameter* menggunakan metode *Grid Search* serta penggunaan regularisasi *dropout*. Penggunaan regularisasi *dropout* dalam penelitian ini memiliki peran penting dalam mencegah *overfitting* serta

meningkatkan kemampuan model untuk menggeneralisasi data yang belum pernah dilihat sebelumnya.

2.3 *Malware*

Malware (malicious software) adalah perangkat lunak berbahaya yang dirancang untuk merusak sistem. Beberapa tanda dari mesin yang terinfeksi adalah performa yang lambat, pesan *error*, kurangnya ruang penyimpanan, *crash*, jendela *pop-up* yang tidak terduga, *file* hilang, peningkatan aktivitas jaringan, pengalihan situs web ke situs web yang aneh, keamanan komputer yang dinonaktifkan, dan sistem tidak dapat beroperasi [24] [25]. *Malware* dapat ditransmisikan dalam beberapa metode, termasuk melalui *e-mail* dengan lampiran *file* yang terinfeksi atau melalui pengguna yang mengklik tautan berbahaya.

2.4 *Benign*

Benign adalah istilah yang digunakan untuk mengacu pada *software* atau aplikasi yang tidak berbahaya atau tidak memiliki efek negatif pada sistem komputer [26]. *Benign* adalah *software* yang tidak memiliki tujuan jahat seperti melakukan peretasan, mengumpulkan informasi rahasia, atau menyebarkan virus. *Benign* memiliki tujuan positif seperti membantu pengguna menyelesaikan tugas atau berkomunikasi. Contoh *benign* adalah *software* pemrosesan teks, browser web, atau aplikasi pemutar musik.

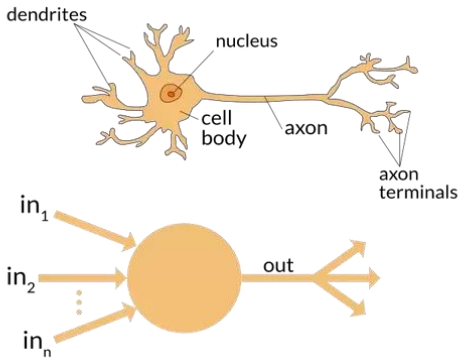
2.5 *Machine Learning*

Machine learning (ML) yang merupakan bagian dari *artificial intelligence (AI)*, telah berkembang secara signifikan dalam beberapa tahun terakhir dalam konteks analisis data dan komputasi, yang seringkali memungkinkan aplikasi beroperasi dengan cara yang cerdas [27]. *Machine learning* berkaitan erat dengan statistik komputasi, *data mining and analytics*, *data science*, terutama berfokus pada pembuatan sistem/mesin untuk belajar dan memperoleh pengetahuan secara otomatis dengan mengekstraksi pola dari *raw data* [28] [29] [30]. Dengan demikian, model *machine learning* biasanya terdiri dari kumpulan pedoman, metode, atau

"fungsi transfer" kompleks yang dapat diterapkan untuk menemukan pola data yang menarik seperti untuk mendeteksi perilaku atau lainnya, yang dapat menjadi sangat penting dalam bidang keamanan siber [31].

2.6 *Deep Learning* dan *Neural Network*

Deep learning adalah bagian dari *machine learning* di bidang *artificial intelligence (AI)*, yang merupakan model komputasi yang terinspirasi dari jaringan saraf biologis di otak manusia [29]. Otak manusia terdiri dari sel-sel yang disebut sebagai neuron dalam jaringan saraf. Demikian pula, di otak manusia, semua sel terhubung melalui akson dan dendrit dengan wilayah koneksi yang dikenal sebagai sinapsis. Sambungan-sambungan ini bila ditemukan pada *Artificial Neural Network (ANN)*, mengandung bobot-bobot untuk berperilaku sebagai sambungan antar sel saraf pada otak manusia [32]. Gambar 2.1 menunjukkan otak manusia dan versi simulasi otak manusia melalui *artificial neural network* [32].



Gambar 2.1 Otak manusia dan simulasinya melalui *ANN* [32]

Perbedaan utama antara *deep learning* dan *machine learning* konvensional adalah kinerjanya saat volume data meningkat. Algoritma *deep learning* biasanya bekerja dengan baik ketika volume data besar, sedangkan algoritma *machine learning* bekerja lebih baik pada *dataset* yang lebih kecil [33]. Pendekatan *deep learning* meniru mekanisme otak manusia untuk menginterpretasikan sejumlah besar data atau data kompleks seperti gambar, suara, dan teks [33] [34]. Dalam hal ekstraksi fitur untuk membangun model, *deep learning* mengurangi upaya yang diperlukan untuk merancang ekstraktor fitur dibandingkan dengan teknik *machine*

learning konvensional [35]. Selain karakteristik ini, *deep learning* biasanya membutuhkan waktu lama untuk melatih suatu algoritma daripada algoritma *machine learning* [33]. Oleh sebab itu, *deep learning* lebih bergantung pada mesin berperforma tinggi yang dilengkapi dengan *GPU* daripada algoritma *machine learning* konvensional [33] [36].

2.7 *Back Propagation Neural Network*

Neural Network (NN) biasanya diklasifikasikan menjadi beberapa jenis berdasarkan metode *supervised* dan *unsupervised learning* serta arsitektur *feed-forward* dan *feed-backward* [37]. *Back-Propagation Neural Network* (*BPNN*) adalah salah satu model *Neural Network (NN)* yang menggunakan metode *supervised-learning* dan arsitektur *feed-forward* yang diusulkan oleh Rumelhart et al. [38], *BPNN* belajar dengan menghitung *error* pada *output layer* untuk menemukan *error* pada *hidden layer* [39].

BPNN adalah salah satu model *Neural Network (NN)* yang paling sering digunakan untuk klasifikasi dan prediksi [40]. Dengan *backpropagation*, data *input* terkait berulang kali disajikan ke jaringan saraf. Dalam setiap iterasi, *output* jaringan saraf dibandingkan dengan *output* yang diinginkan, dan *error* dihitung. *Error* ini kemudian di-*backpropagated* ke jaringan saraf dan digunakan untuk menyesuaikan bobot guna mengurangi *error* di setiap iterasi dan mendekatkan model *neural* untuk menghasilkan *output* yang diinginkan [40].

2.7.1 Algoritma Pembelajaran *Back Propagation*

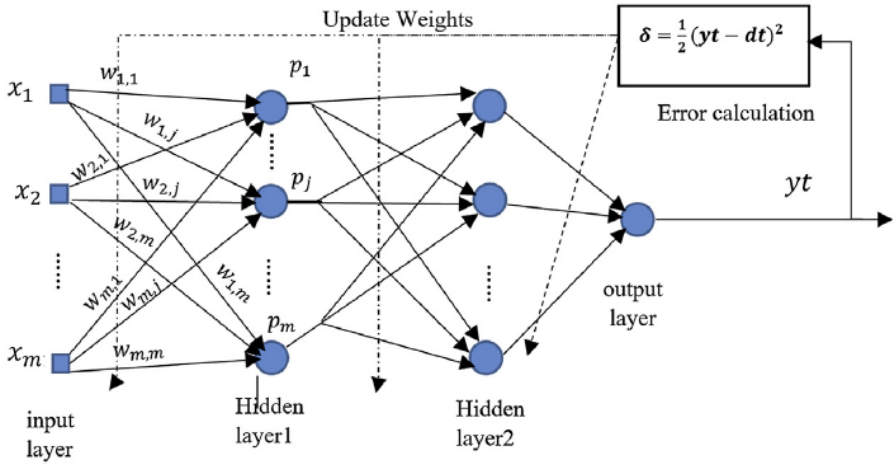
Berikut adalah gambaran proses pembelajaran *Back Propagation* [41]:

1. *Forward propagation of operating signal* (propagasi maju dari sinyal operasi) : sinyal *input* bergerak dari *input layer* ke *output layer* melalui *hidden layer*. Nilai bobot dan nilai bias jaringan dipertahankan konstan selama propagasi maju dari sinyal operasi dan keadaan setiap neuron *layer* hanya akan memengaruhi neuron *layer* berikutnya. Jika *output* yang diharapkan tidak dapat dicapai pada *output layer*, maka dapat dialihkan ke propagasi balik sinyal *error*.

2. *Back Propagation of error signal* (propagasi balik dari sinyal *error*): sinyal *error* adalah perbedaan antara *output* nyata dan *output* yang diharapkan dari jaringan. Dalam propagasi balik sinyal *error*, sinyal *error* ditransmisikan dari ujung *output* ke *input layer* dengan cara *layer* demi *layer*. Selama propagasi balik sinyal *error*, nilai bobot jaringan diatur oleh umpan balik *error*. Untuk mendapatkan *output* nyata dari jaringan yang lebih dekat dengan yang diharapkan, nilai bobot dan nilai bias secara kontinyu terus dimodifikasi.

2.7.2 Deskripsi Algoritma *Back Propagation* dalam Matematika

Backpropagation didasarkan pada formulasi yang lebih menyeluruh dari aturan rantai dari kalkulus. Aturan rantai untuk turunan terurut adalah redefinisi yang memungkinkan seseorang untuk mempertimbangkan beberapa level jaringan saraf [42]. Singkatnya, lihat Gambar 2.2 yang menunjukkan *multilayer perceptron* sederhana dengan satu atau lebih *hidden layer* [42].



Gambar 2.2 *Backpropagation* dalam *multilayer perceptron* dengan dua *hidden layer*, menunjukkan koneksi sinaptik antar neuron di *layer* yang berbeda [42]

Dapat digambarkan sebagai [42]:

$$p^{n+1}(i) = \sum_{j=1}^{m} w^{n+1}(i,j)x^{n}(i) + b^{n+1}(i) \qquad 2.1$$

Dimana: p adalah net *input* ke neuron, m adalah jumlah *input* ke neuron, n adalah step waktu saat ini, w adalah bobot *input* yang sesuai, b adalah bias neuron, i adalah indeks neuron dan j adalah indeks *input*.

Output dari neuron i adalah:

$$y^{n+1}(i) = f^{n+1}\left(p^{n+1}(i)\right) \quad 2.2$$

Dimana y adalah keluaran neuron, f adalah fungsi aktivasi dari neuron ke-i.

Error total δ dihitung dengan membandingkan keluaran perceptron yt dengan keluaran yang diinginkan $\mathrm{d(t)}$ seperti pada 2.3:

$$\delta = \frac{1}{2}\left(y(t) - d(t)\right)^2 \quad 2.3$$

Memperkecil *error* menggunakan penurunan gradien dicapai dengan menghitung turunan parsial δ sehubungan dengan setiap bobot dalam jaringan. Turunan parsial untuk *error* dihitung dalam dua tahap: maju, seperti yang dijelaskan dalam 2.1 dan 2.2, dan mundur, dimana turunannya di-backpropagated dari *output layer* kembali ke *input layer*. Tahap mundur dimulai dengan menghitung $\frac{\partial\delta}{\partial \mathrm{y}}$untuk setiap unit keluaran. Diferensiasi 2.3 untuk pola *input* spesifik n menghasilkan 2.4:

$$\frac{\partial\delta}{\partial y_n} = y_n - d_n \quad 2.4$$

Menerapkan aturan rantai untuk menghitung turunan *error* sehubungan dengan *input* $\frac{\partial\delta}{\partial \mathrm{x_n}}$:

$$\frac{\partial\delta}{\partial x_n} = \frac{\partial\delta}{\partial y_n} \times \frac{dy_n}{dx_n} \quad 2.5$$

Nilai $\frac{\mathrm{dy}}{\mathrm{dx}}$ diperoleh dari perkalian 2.2. Substitusi 2.5 menghasilkan :

$$\frac{\partial E}{\partial x_i} = \frac{\partial E}{\partial y_i} \times f'(p_i) \quad 2.6$$

Ini menunjukkan bagaimana perubahan *input* x akan mempengaruhi *error*. Karena *input* total adalah fungsi linier dari bobot pada koneksi, mudah untuk menghitung bagaimana *error* akan dipengaruhi oleh perubahan status *input* dan bobot. Turunan parsial dari *error* sehubungan dengan bobot dapat didefinisikan oleh:

$$\frac{\partial E}{\partial w_y} = \frac{\partial E}{\partial x_i} \times \frac{\partial x_i}{\partial w_{ij}} = \frac{\partial E}{\partial x_i} \times y_j \quad 2.7$$

Output dari setiap neuron berkontribusi pada $\frac{\partial \mathrm{E}}{\partial \mathrm{y_j}}$ dihasilkan dari neuron i pada j adalah:

$$\frac{\partial E}{\partial x_i} \times \frac{\partial x_i}{\partial y_j} = \frac{\partial E}{\partial x_i} \times w_{ij} \quad 2.8$$

Dari 2.8 rumus umum untuk semua koneksi ke unit j dapat dihasilkan:

$$\frac{\partial E}{\partial y_j} = \sum_i \frac{\partial E}{\partial x_i} \times w_{ij} \quad 2.9$$

Turunan parsial dari *error* sehubungan dengan bobot digunakan untuk mengubah bobot setelah setiap pola *input-output*, yang tidak memerlukan memori khusus untuk turunan itu sendiri disebut *online training*. Pendekatan alternatif adalah mengakumulasi kesalahan $\frac{\partial \mathrm{E}}{\partial \mathrm{w}}$ pada semua pasangan *input-output* dalam set *training* sebelum memperbarui bobot yang sesuai. Pendekatan ini juga dikenal sebagai offline (batch) *training*. Versi paling sederhana dari penurunan gradien adalah mengubah setiap bobot dengan jumlah yang sebanding dengan akumulasi *error* seperti yang dijelaskan dalam 2.10:

$$\Delta w = {}^{-\varepsilon \partial E}/_{\partial w} \quad 2.10$$

Sebagai penyempurnaan metode ini, digunakan metode percepatan di mana gradien arus digunakan untuk mengubah kecepatan titik dalam ruang w alih-alih posisinya:

$$\Delta w(n) = \frac{-\varepsilon \partial E}{\partial w} + \alpha \Delta w(n-1) \quad 2.11$$

di mana n adalah *epoch's integer index*, a adalah *learning rate* atau *decay factor* yang menentukan rasio kontribusi riwayat *training* dan gradien arus terhadap perubahan bobot, $\propto \in (0,1]$.

2.8 Fungsi Aktivasi

Fungsi Aktivasi merupakan elemen penting dalam *neural network* yang berfungsi untuk mengintegrasikan *input* dari neuron sebelumnya dan menghasilkan *output* yang akan diteruskan ke neuron berikutnya [43]. Fungsi ini menentukan nilai keluaran berdasarkan nilai total masukan pada neuron. Fungsi aktivasi suatu algoritma befungsi untuk menentukan apakah neuron tersebut harus "aktif" atau tidak berdasarkan dari *weight*ed sum dari *input*.

Fungsi aktivasi dapat dibedakan menjadi dua jenis, yaitu linier dan non-linier, tergantung pada fungsi yang merepresentasikannya. Fungsi aktivasi linier menghasilkan keluaran yang sebanding dengan masukan, sedangkan fungsi aktivasi non-linier menghasilkan keluaran yang tidak sebanding dengan masukan. Fungsi aktivasi linier digunakan pada *neural network* sederhana seperti perceptron, sedangkan fungsi aktivasi non-linier digunakan pada *neural network* yang lebih kompleks seperti *multilayer perceptron*.

Fungsi aktivasi non-linier digunakan untuk mengatasi keterbatasan model linier, sehingga jaringan saraf mampu memodelkan data yang lebih kompleks dan memahami hubungan non-linear di antara fitur-fitur yang ada. Fungsi ini menjadikan *neural network* lebih efektif dalam memecahkan berbagai masalah, seperti klasifikasi, regresi, dan tugas-tugas lainnya. Tanpa fungsi aktivasi, jaringan saraf akan menjadi sekumpulan operasi linear, sehingga keuntungan dari struktur *layer* dalam model tidak dapat dimaksimalkan. Beberapa contoh fungsi aktivasi non-linier yang umum digunakan adalah *sigmoid*, *softmax*, *ReLU*, dan *tanh*.

2.8.1 *Sigmoid Function*

Fungsi aktivasi *sigmoid* adalah salah satu jenis fungsi aktivasi yang umum digunakan dalam *neural network*. Fungsi ini mengubah *input*nya menjadi nilai probabilitas dalam rentang 0 hingga 1. [43]. Bentuk matematis dari fungsi *sigmoid* adalah

$$f(x) = \frac{1}{1 + e - x} \tag{2.12}$$

di mana x is *input* dari neuron tersebut, and $f(x)$ adalah *output* dalam rentang [0, 1]. Ketika *input x* positif, *output* $f(x)$ cenderung menjadi 1, dan ketika x negatif, *output* $f(x)$ cenderung menjadi 0. Sifat ini membuat fungsi *sigmoid* cocok digunakan dalam masalah klasifikasi biner, di mana tujuannya adalah mengklasifikasikan data ke dalam salah satu dari dua kelas yang berbeda.

2.8.2 *Softmax Function*

Fungsi aktivasi *softmax* adalah fungsi aktivasi yang umum digunakan dalam lapisan *output* dari *neural network*, terutama pada tugas klasifikasi multikelas. Fungsi ini mengubah nilai *input*nya menjadi distribusi probabilitas, di mana setiap nilai *output* mewakili probabilitas prediksi untuk masing-masing kelas yang ada [43].

Rumus matematika dari fungsi *softmax* adalah:

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \quad 2.13$$

dengan xi adalah nilai *input* untuk kelas ke-i, e adalah bilangan konstan Euler, i,j = indeks unit dan lapisan dan K adalah jumlah total kelas dalam tugas klasifikasi.

Kelebihan utama dari *softmax* activation function adalah kemampuannya untuk menghasilkan probabilitas prediksi yang dapat digunakan untuk memutuskan kelas yang paling mungkin untuk contoh data tertentu dalam klasifikasi multikelas. *Output* dari fungsi ini berada dalam rentang 0 hingga 1, dan total probabilitas untuk semua kelas akan sama dengan 1, sehingga memudahkan dalam interpretasi dan pemilihan kelas dengan probabilitas tertinggi [44].

2.8.3 *ReLU Function*

Fungsi aktivasi *ReLU (Rectified Linear Unit)* adalah fungsi aktivasi yang populer dan sering digunakan dalam *neural network*. Fungsi ini akan memberikan keluaran 0 ketika $x<0$ dan merepresentasikan fungsi linier ketika $x \geq 0$. Dalam bentuk matematis, rumus *ReLU* dapat dinyatakan sebagai:

$$f(x) = max(0, x) = \{xi, if\ xi \geq 0;\ 0, if\ xi < 0 \quad 2.14$$

Kelebihan utama dari *ReLU* activation function adalah sifat non-linear dan kemampuannya dalam mengatasi masalah gradien menghilang (*vanishing gradient problem*). Karena fungsi ini sederhana dan hanya memiliki dua cabang, yaitu 0 untuk *input* negatif dan *x* untuk *input* positif, perhitungan gradien pada lapisan yang menggunakan *ReLU* menjadi lebih mudah dan efisien [44] [45]. Sehingga, *ReLU* telah membantu mempercepat proses *training* dan meningkatkan performa model jaringan saraf, terutama pada *deep neural network*.

2.8.4 *Tanh (Hyperbolic Tangent) Function*

Fungsi aktivasi *Tanh (Hyperbolic Tangent)* adalah fungsi aktivasi yang sering digunakan dalam jaringan saraf tiruan. Fungsi ini mirip dengan fungsi aktivasi *sigmoid*, namun memiliki rentang *output* dari -1 hingga 1, sehingga *outputnya zero-centered*. Dalam bentuk matematis, rumus Tanh dapat dinyatakan sebagai:

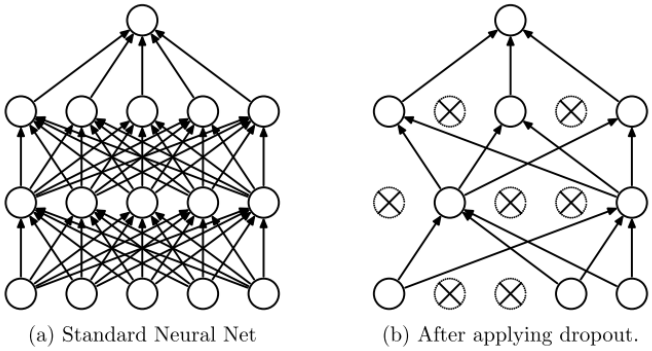
$$f_{tanh}(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.15}$$

dengan x sebagai nilai *input*.

Kelebihan utama dari *Tanh function* adalah kemampuannya untuk mengatasi masalah *non-zero-centered* yang dimiliki oleh *sigmoid*. Hal ini dapat membantu meningkatkan konvergensi pada saat melatih jaringan saraf dan mengurangi masalah gradien menghilang. Selain itu, karena rentang *output*nya dari -1 hingga 1, Tanh juga dapat membantu mempercepat proses pembelajaran dibandingkan dengan *sigmoid*, terutama ketika data *input* memiliki skala yang besar.

2.9 Regularisasi *Dropout*

Dropout adalah teknik regularisasi yang umum digunakan dalam jaringan saraf tiruan untuk mengurangi *overfitting*. Teknik ini diperkenalkan oleh Geoffrey Hinton dan rekan-rekannya pada tahun 2012 sebagai cara untuk mencegah jaringan saraf menjadi terlalu bergantung pada neuron-neuron tertentu selama proses *training* [46]. Ide utama di balik *dropout* adalah dengan secara acak mengabaikan atau menggugurkan (*drop*) sebagian unit atau neuron beserta koneksi-koneksi mereka dalam jaringan selama *training*. Hal ini mencegah terjadinya ketergantungan yang berlebihan antar unit dan mendorong jaringan untuk mempelajari fitur-fitur yang lebih kuat dan generalisasi yang lebih baik. Dengan mengabaikan unit-unit tertentu, jaringan menjadi kurang sensitif terhadap keberadaan setiap neuron individu dan belajar untuk mengandalkan pengetahuan kolektif dari beberapa neuron [46].



Gambar 2.3 *Dropout* model *neural network* (a) Standar *neural network* dengan 2 *hidden layer*, (b) Contoh penerapan *dropout* pada network, unit yang disilangkan telah di-*drop* [46]

Pada saat *training*, setiap unit dalam jaringan, termasuk koneksi-koneksi mereka, memiliki probabilitas untuk di-*dropout* atau dinonaktifkan. Probabilitas ini biasanya diatur antara 0,1 hingga 0,5 namun probabilitas yang spesifik dapat dipilih berdasarkan kebutuhan dan karakteristik data serta model yang digunakan [46]. Unit-unit yang di-*dropout* tidak berkontribusi pada proses *forward pass* atau *backward pass* jaringan selama *training*, sehingga secara efektif mengurangi kapasitas jaringan dan mencegah terjadinya *overfitting* [46].

Teknik *dropout* telah terbukti meningkatkan performa jaringan saraf pada berbagai tugas, termasuk pengenalan gambar, pengenalan ucapan, dan klasifikasi dokumen [46]. *Dropout* juga telah mencapai hasil terbaik pada banyak *dataset* referensi [46]. Dengan mencegah *overfitting*, *dropout* memungkinkan jaringan saraf untuk menggeneralisasi lebih baik pada data yang belum pernah dilihat sebelumnya dan meningkatkan kemampuan jaringan untuk melakukan prediksi yang akurat.

2.10 *Adam Optimizer*

Adam merupakan singkatan dari "*Adaptive Moment Estimation*," dan merupakan teknik optimasi yang populer yang banyak digunakan dalam *training* model *neural network* [47]. Tujuan utama dari *Adam* adalah untuk mengoptimalkan *learning rate* secara otomatis dan efisien selama proses *training* jaringan, sehingga

membantu model untuk mencapai konvergensi yang lebih cepat dan mencapai generalisasi yang lebih baik pada data yang belum pernah dilihat sebelumnya [48] [49].

Adam optimizer memperbarui bobot dan bias berdasarkan gradien dari fungsi *loss* (fungsi yang mengukur seberapa besar kesalahan prediksi jaringan) terhadap parameter-parameter tersebut. *Adam* menggabungkan konsep momentum dan RMSprop untuk mencapai *learning rate* yang adaptif untuk setiap parameter selama proses *training*. Rumus-rumus untuk memperbarui parameter menggunakan *Adam optimizer* adalah sebagai berikut [48]:

1. Rata-rata bergerak dari momen pertama (*mean*) dari gradien (m):

$$m_t = \beta_1 . m_{t-1} + (1 - \beta_1) . g_t \quad (2.16)$$

2. Rata-rata bergerak dari momen kedua (variansi tidak terpusat) dari gradien (v):

$$v_t = \beta_2 . v_{t-1} + (1 - \beta_2) . {g_t}^2 \quad (2.17)$$

3. Estimasi momen pertama yang dikoreksi bias:

$$\widehat{m}_t = \frac{m_t}{1 - {\beta_1}^t} \quad (2.18)$$

4. Estimasi momen kedua yang dikoreksi bias:

$$\hat{v}_t = \frac{v_t}{1 - {\beta_2}^t} \quad (2.19)$$

5. Aturan pembaruan untuk setiap parameter (θ):

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{\hat{v}_{t+\epsilon}}} . \widehat{m}_t \quad (2.20)$$

di mana:

m_t mewakili rata-rata bergerak dari momen pertama gradien pada waktu t.

v_t mewakili rata-rata bergerak dari momen kedua gradien pada waktu t.

g_t mewakili gradien dari fungsi *loss* terhadap parameter pada waktu t.

β_1 and β_2 adalah laju penurunan eksponensial untuk estimasi momen pertama dan kedua, secara berturut-turut (biasanya diatur menjadi 0,9 dan 0,999).

η adalah *learning rate*, yang mengontrol ukuran pembaruan parameter.

ϵ adalah konstanta kecil (biasanya 10^{-8}) yang ditambahkan ke penyebut untuk stabilitas numerik.

2.11 *Cross Entropy Loss Function*

Cross entropy loss function adalah sebuah fungsi *loss* yang umum digunakan dalam *training* model untuk masalah klasifikasi dalam *deep learning*. Fungsi ini berfungsi untuk mengukur seberapa besar perbedaan antara distribusi probabilitas yang dihasilkan oleh model dengan distribusi probabilitas yang sebenarnya dari data yang sedang diolah [44].

Dalam konteks klasifikasi, model akan mengeluarkan probabilitas untuk setiap kelas yang mungkin dari data yang diberikan. *Cross entropy loss function* akan membandingkan probabilitas yang dihasilkan oleh model dengan probabilitas yang benar (*one-hot encoded vector*), dan memberikan nilai *loss* berdasarkan perbedaan antara keduanya. Jika probabilitas yang dihasilkan oleh model mendekati probabilitas yang benar, maka nilai *loss* akan lebih kecil, dan sebaliknya, jika probabilitas yang dihasilkan sangat berbeda dengan probabilitas yang benar, maka nilai *loss* akan lebih besar. Rumus dari *Cross Entropy Loss Function* dapat dituliskan sebagai berikut:

$$H(p,q) = -\sum_i p(i)\, log\, q(i) \tag{2.21}$$

di mana *H(p,q)* adalah nilai *loss* (*cross entropy*) antara distribusi probabilitas *p* (probabilitas yang benar) dan *q* (probabilitas yang dihasilkan oleh model), dan *i* adalah indeks untuk setiap kelas yang mungkin.

2.12 *Grid Search*

Grid Search merupakan salah satu metode yang sering digunakan dalam optimasi *hyperparameter* pada model *machine learning* dan *neural network* [50]. Teknik ini melibatkan eksplorasi sistematis pada kombinasi *hyperparameter* yang telah ditentukan dalam sebuah *grid* untuk mencari kombinasi yang paling optimal yang dapat meningkatkan performa model. Dalam sebuah studi perbandingan tentang teknik optimasi *hyperparameter* untuk *deep learning*, *Grid Search* ditemukan sebagai pendekatan yang berharga dalam mencari nilai *hyperparameter* optimal [50]. Metode ini membantu pengguna, pengembang, analis data, dan peneliti dalam menggunakan dan mengadaptasi model *deep learning* secara efektif.

Kelebihan menggunakan *Grid Search* adalah pendekatannya yang sederhana dan intuitif [51]. Dengan melakukan eksplorasi pada setiap kombinasi *hyperparameter* dalam *grid*, *Grid Search* dapat menemukan kombinasi yang menghasilkan model dengan performa terbaik. Selain itu, *Grid Search* juga membantu memastikan bahwa *hyperparameter* yang diuji telah mencakup variasi yang luas dan mendalam dalam ruang *hyperparameter*. Hal ini dapat meningkatkan peluang untuk menemukan kombinasi *hyperparameter* yang optimal dan menghindari masalah *overfitting* atau *underfitting* pada model [51].

Meskipun *Grid Search* memiliki kelebihan dalam kemampuan eksplorasi dan pendekatan yang sederhana, ada juga beberapa keterbatasan. Salah satu keterbatasan utama dari *Grid Search* adalah kinerjanya yang komputasional tinggi, terutama ketika jumlah *hyperparameter* yang diuji atau nilai yang diuji sangat besar [52]. Proses eksplorasi pada *grid hyperparameter* memerlukan waktu dan sumber daya komputasi yang signifikan [52] [53]. Oleh karena itu, *Grid Search* mungkin tidak cocok untuk digunakan pada *dataset* yang sangat besar atau model yang kompleks.

2.13 *Confusion Matrix*

Confusion matrix merupakan sebuah alat evaluasi kinerja yang penting dalam dunia pemodelan dan analisis klasifikasi. *Confusion matrix* ini umumnya digunakan untuk mengevaluasi keakuratan dan performa suatu model klasifikasi dengan membandingkan hasil prediksi model terhadap data aktual. Secara sederhana, *confusion matrix* menggambarkan jumlah data yang diklasifikasikan dengan benar maupun salah ke dalam kategori positif dan negatif [54].

Confusion matrix terdiri dari empat komponen utama, yaitu *True Positive (TP), True Negative (TN), False Positive (FP),* dan *False Negative (FN). True Positive* mencerminkan jumlah data positif yang diklasifikasikan dengan benar oleh model, sedangkan *True Negative* mengindikasikan jumlah data negatif yang diklasifikasikan dengan benar. Di sisi lain, *False Positive* merupakan jumlah data negatif yang salah diklasifikasikan sebagai positif, dan *False Negative* menunjukkan jumlah data positif yang salah diklasifikasikan sebagai negatif [54].

Tabel 2.2 *Confusion matrix*

		Predicted Values	
		Negative	*Positive*
Actual Values	*Negative*	TN	FP
	Positive	FN	TP

Confusion matrix dapat digunakan untuk menghitung beberapa metrik kinerja model klasifikasi, yaitu sebagai berikut:

a. Akurasi (*Accuracy*): Merupakan rasio jumlah prediksi yang benar (*TP* dan *TN*) dengan keseluruhan data. Metrik ini mengukur sejauh mana model dapat memprediksi dengan tepat.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad 2.22$$

b. Presisi (*Precision*): Menghitung seberapa banyak dari data yang diprediksi positif oleh model yang *True Positive (TP)*. Metrik ini membantu dalam mengukur ketepatan model dalam mengklasifikasikan data sebagai positif.

$$Precision = \frac{TP}{TP + FP} \quad 2.23$$

c. *Recall* atau *True Positive Rate*: Mengukur kemampuan model untuk mengenali semua contoh positif yang sebenarnya. *Recall* dihitung dengan membagi TP dengan jumlah keseluruhan contoh positif.

$$Recall/TPR = \frac{TP}{TP + FN} \quad 2.24$$

d. *F1-score*: Membantu dalam menemukan keseimbangan antara presisi dan *recall* ketika terdapat ketidakseimbangan kelas.

$$F1 - score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad 2.25$$

2.14 Python

Python adalah bahasa pemrograman tingkat tinggi dan interpretatif yang dirancang untuk menjadi jelas, ringkas, dan ekspresif. Python dirilis pada tahun 1991 oleh Guido van Rossum, dan sejak saat itu, Python telah menjadi salah satu bahasa pemrograman paling populer di dunia. Python adalah bahasa pemrograman yang serbaguna yang dapat digunakan untuk berbagai aplikasi, termasuk analisis data, *machine learning*, dan pengembangan web [55]. Python memiliki sejumlah keunggulan, yang membuat Python menjadi bahasa pemrograman yang populer. Keunggulan-keunggulan tersebut antara lain:

a. Python memiliki sintaks yang sederhana dan mudah dibaca, sehingga Python mudah dipelajari oleh pemula.
b. Python memiliki beragam *library* dan *framework* yang dapat digunakan untuk berbagai tugas, sehingga Python dapat digunakan untuk mengembangkan berbagai jenis aplikasi.
c. Python memiliki komunitas yang aktif dan besar, sehingga Python memiliki dokumentasi dan sumber daya yang tersedia secara luas.

Python telah mengembangkan beragam *library* dan *framework* yang sangat bermanfaat dalam bidang *machine learning*. Misalnya, scikit-learn, TensorFlow, dan PyTorch adalah beberapa contoh populer yang sering digunakan dalam mengembangkan *machine learning*. Scikit-learn adalah *library* Python yang efisien dan mudah digunakan, menyediakan berbagai algoritma *machine learning* untuk tugas-tugas seperti klasifikasi, regresi, dan *clustering* [56]. Selain itu, untuk tugas-tugas yang lebih kompleks seperti *deep learning*, TensorFlow dan PyTorch menjadi pilihan utama. TensorFlow dan PyTorch adalah *framework deep learning* yang populer, memiliki dukungan *GPU* yang kuat, dan memungkinkan para peneliti untuk mengembangkan arsitektur *neural network* yang canggih.

Keunggulan dari *library-library* ini terletak pada penerapan metode-metode yang sudah terbukti dan memiliki performa tinggi dalam *machine learning*. Dalam pengembangan model, para *data scientist* dan *engineer machine learning* dapat dengan mudah mengimplementasikan algoritma regresi, klasifikasi, dan *clustering*

untuk memecahkan berbagai permasalahan [56]. *Library-library* ini juga menyediakan fitur-fitur seperti *preprocessing* data untuk mempersiapkan data sebelum diproses, evaluasi model untuk mengukur kinerja model, serta visualisasi hasil untuk memahami dan menyajikan informasi yang dihasilkan oleh model.

Gambar 2.4 Logo bahasa pemrograman Python [57]

2.15 Flask Python

Flask adalah *micro framework* untuk pengembangan aplikasi web berbasis Python. Flask memungkinkan *developer* untuk membangun aplikasi web dengan cara yang fleksibel dan efisien dengan hanya menambahkan modul dan fungsi yang diperlukan. Flask dikembangkan oleh Armin Ronacher, yang memimpin tim penggemar Python internasional bernama Poocco. Flask didasarkan pada *toolkit* Werkzeg *WSGI (Web Server Gateway Interface)* dan mesin *template* Jinja2. Keduanya adalah proyek Pocco [58].

Gambar 2.5 Logo Flask [59]

2.16 Google Colaboratory

Google Colaboratory, yang sering disebut sebagai Colab, adalah sebuah platform berbasis *cloud* yang disediakan oleh Google untuk keperluan pengembangan dan eksekusi kode pemrograman dalam bahasa Python [60]. Platform ini memungkinkan para pengguna, terutama para ilmuwan data, peneliti, dan pengembang perangkat lunak, untuk melakukan analisis data, pemrosesan gambar, *training* model *machine learning*, serta penelitian dan pengembangan

berbagai proyek teknis lainnya melalui browser web. Colab menawarkan lingkungan komputasi yang kuat dengan dukungan akses ke *Graphics Processing Unit (GPU)* dan *Tensor Processing Unit (TPU)*, yang dapat sangat meningkatkan kecepatan *training* model *machine learning* dan pemrosesan data yang intensif [60] [61].

Colab menyediakan notebook interaktif yang memungkinkan pengguna untuk menggabungkan kode, teks, dan elemen visual dalam satu dokumen. Setiap notebook berjalan di lingkungan virtual yang di-*hosting* di *cloud*, yang berarti pengguna tidak perlu khawatir tentang konfigurasi perangkat keras atau perangkat lunak. Colab juga mendukung berbagai *library* populer seperti TensorFlow, Keras, PyTorch, dan lainnya, yang memudahkan pengguna dalam melakukan eksperimen dan pengembangan dengan *framework machine learning* [60].

Gambar 2.6 Logo Google Colaboratory [62]

2.17 PyCharm

PyCharm adalah *Integrated Development Environment* (*IDE*) yang sangat populer yang dikembangkan oleh JetBrains. *IDE* ini dirancang untuk memfasilitasi pengembangan perangkat lunak dengan efisiensi tinggi. PyCharm menawarkan berbagai fitur lengkap yang mendukung seluruh proses pengkodean, *debugging*, dan pengujian aplikasi Python [63]. *IDE* ini memiliki antarmuka yang intuitif dan *user-friendly*, serta berbagai alat bantu yang signifikan dalam meningkatkan produktivitas para pengembang.

PyCharm menyediakan kemampuan pengkodean otomatis, pemeriksaan kode, dan fitur *refactor* yang secara signifikan membantu mengurangi kesalahan serta meningkatkan kualitas kode. Selain itu, *IDE* ini juga mendukung pengembangan berbasis web dengan menyediakan alat untuk pengembangan *front-*

end dan *back-end*. Fitur integrasi dengan sistem kontrol versi seperti Git secara efektif mempermudah pengelolaan kode sumber [63].

Gambar 2.7 Logo PyCharm [64]

2.18 Anaconda

Anaconda adalah platform yang sangat berguna untuk pengembangan dan analisis data dalam bahasa pemrograman Python. Platform ini mencakup berbagai *tools* dan *library* yang dirancang untuk memudahkan pengelolaan lingkungan pengembangan, pengujian, dan penerapan kode Python, serta analisis data yang kompleks. Anaconda menyediakan manajemen paket yang efisien melalui *tools* bernama "conda", yang memungkinkan pengguna untuk menginstal, mengelola, dan memperbarui *library-library* dan dependensi yang diperlukan dalam proyek. Keuntungan lain dari Anaconda adalah kemampuannya dalam membuat dan mengelola lingkungan virtual yang terpisah, sehingga pengguna dapat mengembangkan proyek dengan *library* dan versi yang berbeda-beda tanpa konflik [65].

Gambar 2.8 Logo Anaconda [66]