PENYELESAIAN PERMAINAN *RUBIK'S CUBE MENGGUNAKAN NEURAL NETWORK BACKPROPAGATION*

Taufik Setiyadi

Tehnik Informatika Universitas Komputer Indonesia
Jl.Dipatiukur 112-114 Bandung
E-mail : taufik.setiyadi@gmail.com

ABSTRAK

Rubik's CUBE adalah sebuah permainan *puzzle* mekanik tiga dimensi yang ditemukan pada tahun 1974 oleh seorang pemahat dan profesor arsitektur dari *Hungaria* bernama *Erno Rubik*. *Rubik's CUBE* pertama berukuran 3x3x3 berbentuk kubus kecil. Tiap sisi kubus kecil yang mengahadap keluar memiliki warna yang berbeda. *Rubik's CUBE* ini terdiri atas dua puluh enam kubus kecil yang disatukan oleh sebuah kubus kecil di pusat *Rubik's CUBE*. Setiap sisi *Rubik's CUBE* dapat diputar sebanyak dan searah maupun berlawana arah jarum jam.

Rubik's CUBE berbentuk kubus yang terdiri dari 26 kubus kecil yang dapat berputar pada porosnya. Setiap sisi dari kubus ini memiliki 9 permukaan yang terdiri dari enam warna yang berbeda dan diselesaikan dengan melakukan serangkaian langkah memutar kolom dan baris untuk menyeragamkan warna pada setiap sisinya setelah diacak. Dibutuhkan Algoritma yang dapat memberikan solusi untuk dapat menyelesaikan permainan *Rubik;s CUBE*. Pada penelitian ini Algoritma *Neural Network Backpropagation* akan digunakan untuk memperdiksikan langkah yang akan diambil.

Berdasarkan dari hasil penelitian dapat disimpulkan Implementasi *Neural Network Backpropagation* pada penyelesaian permainan *Rubik's CUBE* dapat memberikan solusi untuk menyelesaikan permainan dan Hasil pengujian dapat diketahui nilai akurasi Algoritma *Backpropagation* mencapai rata-rata 95% benar dalam pengujian dan 5% salah yang diimplementasikan permainan *Rubik's CUBE* 2x2.

Kata kunci : *Computer Vision*, Permainan, *Rubik's CUBE, Neural Netowrk Backpropagation*

1. PENDAHULUAN

Rubik's CUBE adalah sebuah permainan *puzzle* mekanik tiga dimensi yang ditemukan pada tahun 1974 oleh seorang pemahat dan profesor arsitektur dari *Hungaria* bernama *Erno Rubik*. *Rubik's CUBE* pertama berukuran 3x3x3 berbentuk kubus kecil. Tiap sisi kubus kecil yang mengahadap keluar memiliki warna yang berbeda. *Rubik's CUBE* ini terdiri atas dua puluh enam kubus kecil yang disatukan oleh sebuah kubus kecil di pusat *Rubik's CUBE*. Setiap sisi *Rubik's CUBE* dapat diputar sebanyak dan searah maupun berlawana arah jarum jam.

Rubik's CUBE berbentuk kubus yang terdiri dari 26 kubus kecil yang dapat berputar pada porosnya. Setiap sisi dari kubus ini memiliki 9 permukaan yang terdiri dari enam warna yang berbeda dan diselesaikan dengan melakukan serangkaian langkah memutar kolom dan baris untuk menyeragamkan warna pada setiap sisinya setelah diacak. *Puzzle Rubik's CUBE* merupakan satu dari sekian persoalan kombinatorial yang cukup dikenal karena kerumitannya, di mana terdapat 43×10^{18} konfigurasi berbeda yang mungkin dihasilkan dengan mengacaknya.

Penyelesaian *puzzle Rubik's CUBE* dengan mengandalkan intuisi maupun dengan langkah-langkah tertentu yang dihafalkan manusia biasanya memerlukan lebih dari 40 langka. Hal ini mendorong dilakukannya berbagai penelitian untuk mencari solusi optimal yaitu, serangkaian langkah terpendek diantara berbagai kemungkinan langkah alternatif yang ada untuk menyelesaikan suatu konfigurasi acak *Rubik's CUBE* dengan mensimulasikannya pada program komputer dan menerapkan kecerdasan buatan untuk menyelesaikannya.

Beberapa Algoritma telah diajukan untuk menemukan solusi optimal dalam menyelesaikan *puzzle* ini antara lain, Algoritma *Thistlethwaite* oleh *Morwen Thistlethwaite* pada tahun 1981, Algoritma *Two-Phase* oleh *Herbert Kociemba* pada tahun 1992, Algoritma *IDA** dengan *Pattern Database* oleh *Richard E. Korf* pada tahun 1997, serta pengembangan Algoritma *Two-phase* oleh *Kociemba* dan *Thomas Rokicky* pada tahun 2010.

Dalam penelitian yang telah dilakukan permainan *Rubik's CUBE* bisa diselesaikan dengan Algoritma *Greedy* dan *Backtraking*, pada penelitian tersebut diperoleh kesimpulan bahwa keunggulan metode ini kita bisa menerapkan Algoritma tanpa menggunakan komputer sekalipun. Kekurangan dari metode ini adalah tidak diketahuinya performansi dari kedua Algoritma tersebut karena memang tidak diuji menggunakan komputer sehingga saran yang

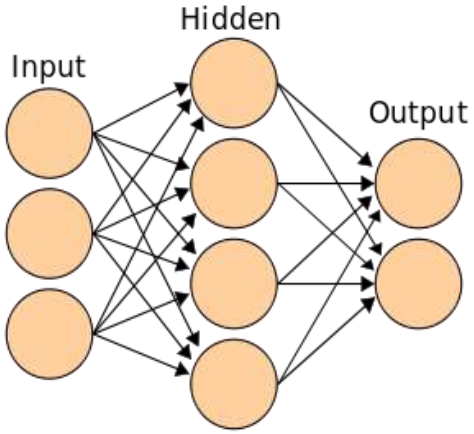
diberikan adalah untuk melakukan percobaan dengan menggunakan Algoritma yang lain yang mungkin lebih bisa memberikan solusi yang lebih bagus.

Salah satu Algoritma yang bisa digunakan adalah Algoritma *Neural Network Backpropagation*. Algoritma ini belum pernah diterapkan untuk menyelesaikan pencarian solusi pada permainan *Rubik's CUBE* dan performansi dalam menyelesaikan masalah tersebut masih belum diketahui. Algoritma *Neural Network Backpropagation* merupakan salah satu kecerdasan buatan yang dapat menyelesaikan pekerjaan dengan diberikan pelatihan. *Neural Network Backpropagation* merupakan jaringan dari sekelompok pemproses elemen, *unit*, atau node-node yang dimodelkan berdasarkan neuron otak. kemampuan pemroses disimpan di dalam kekuatan koneksi antar *neuron* atau bobot (*weights*), kemampuan tersebut diperoleh dari proses adaptasi terhadap atau belajar dari suatu set pola pelatihan.

Berdasarkan latar belakang masalah tersebut maka dapat disimpulkan bagaimana cara mengimplementasikan Algoritma *Neural Network Backpropagation* pada permainan *Rubik's CUBE* untuk mencari solusi pemecahannya dengan tujuan dapat menyelesaikan permainan *Rubik's CUBE* dan untuk mengetahui performansi (waktu) Algoritma *Neural Network Backpropagation* dalam penyelesaikan permainan *Rubik's CUBE*.

1.2 Jaringan Saraf Tiruan

Jaringan saraf tiruan (JST) (Bahasa Inggris A*rtificial Neural Network* (*ANN*) atau juga disebut *Simulated Neural Network* (*SNN*), atau umumnya hanya disebut *Neural Network* (*NN*) adalah jaringan dari sekelompok *unit* pemroses kecil yang dimodelkan berdasarkan jaringan saraf manusia. JST merupakan sistem adaptif yang dapat mengubah strukturnya untuk memecahkan masalah berdasarkan informasi eksternal maupun internal yang mengalir melalui jaringan tersebut. Secara sederhana, JST adalah sebuah alat pemodelan data statistik non-linier. JST dapat digunakan untuk memodelkan hubungan yang kompleks antara input dan *output* untuk menemukan pola-pola pada data diperlihatkan pada gambar dibawah ini.

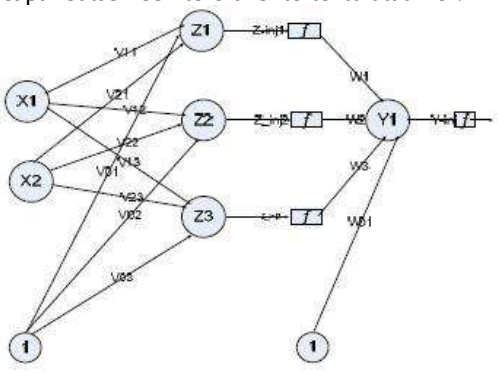


Gambar 1.Pola Input dan Output JST

1.3 Backpropagation

Salah satu arsitektur jaringan saraf tiruan yang sangat populer adalah multilayer *feedforward networks*. Secara umum, jaringan seperti ini terdiri dari sejumlah *unit neuron* sebagai lapisan masukan, satu atau lebih lapisan simpul - simpul *neuron* komputasi lapisan tersembunyi, dan sebuah lapisan simpul - simpul *neuron* komputasi keluaran. Sinyal masukan dipropagasikan ke arah depan (arah lapisan keluaran), lapisan demi lapisan. Jenis jaringan ini adalah hasil generalisasi dari arsitektur *perceptron* satu lapisan, jadi biasa disebut sebagai multilayer *perceptron* (*MLPs*).

Error Backpropagation adalah Algoritma *MLPs* yang menggunakan prinsip pembelajaran terawasi. Propagasi balik (ke arah lapisan masukan) terjadi setelah jaringan menghasilkan keluaran yang mengandung *error*. Pada fase ini seluruh bobot *synaptic* (yang tidak memiliki aktivasi nol) dalam jaringan akan disesuaikan untuk mengkoreksi / memperkecil *error* yang terjadi (*error correction rule*). Untuk pelatihan jaringan, pasangan fase propagasi ke depan dan balik dilakukan secara berulang untuk satu set data latihan, kemudian diulangi untuk sejumlah *Epoch* (satu sesi lewatan untuk seluruh data latihan dalam sebuah proses pelatihan jaringan) sampai *error* yang terjadi mencapai batas kecil toleransi tertentu atau nol.



Gambar 2.Bentuk Jaringan *Backpropagation*

Dapat dilihat bahwa yang bertindak sebagai dendrit adalah X1 dan X2, yaitu data masukan pada jaringan tersebut. Terdapat 2 sinapsis atau bobot yaitu V dan W, sedangkan Z dan Y merupakan bagian dari *soma* atau badan sel dari jaringan tersebut. Dan yang bertindak sebagai akson atau data keluaran adalah Y. Berikut ini Algoritma pembelajaran metode *Backpropagation*.

1. Menentukan Nilai Bobot

Langkah 0 :

Pemberian inisialisasi penimbang (diberi nilai kecil secara acak)

Langkah 1 :

Jika kondisi berhenti tidak tercapai, Ulangi langkah 2 hingga 9 sampai kondisi akhir iterasi dipenuhi(kondisi berhenti jika perulangan mencapai maksimal perulangan atau M*ean Square Error* (*MSE*) sudah lebih kecil dari rasio pembelajaran)

Langkah 2 :

Untuk masing-masing pasangan data pelatihan (*Training* data) lakukan langkah 3 hingga 8

2. Propagasi maju (*Feedforward*)

Langkah 3 :

Masing-masing *unit* masukan (Xi, I = 1,…,n) menerima sinyal masukan Xi dan sinyal tersebut disebarkan ke *unit-unit* bagian berikutnya (*unit-unit* lapisan tersembunyi)

Langkah 4 :

Menghitung nilai *Hidden* dan *Net*, Untuk menghitung nilai *Net*, *input*an harus dinormalisasi dulu dengan menggunakan rumus *sigmoid bipolar*, harus dinormalisasi dengan menggunakan persamaan

$$\boldsymbol{y = f(x) = \frac{1-e^{-x}}{(1+e^{-x})}} \quad \textbf{(2-1)}$$

Masing-masing *unit* dilapisan tersembunyi dikalikan dengan faktor penimbang dan dijumlahkan serta ditambahkan dengan biasnya :

$$\boldsymbol{Z_{in_j} = V_{oj} + \sum_{I=1}^{n} X_i V_{ij}} \quad \textbf{(2-2)}$$

Kemudian menghitung sesuai dengan fungsi aktifasi yang digunakan :

$$\boldsymbol{Z_j = f(Z_{in_j})} \quad \textbf{(2-3)}$$

Bila yang digunakan adalah fungsi *sigmoid* maka bentuk fungsi tersebut adalah

$$\boldsymbol{Z_j = \frac{1}{1+exp^{-z_in_j}}} \quad \textbf{(2-4)}$$

Masing-masing *unit* keluaran (Yk, k=1,2,3,…m) dikalikan dengan faktor penimbang dan dijumlahkan :

$$\boldsymbol{Y_{in_k} = W_{ok} + \sum_{j=1}^{p} Z_j W_{jk}} \quad \textbf{(2-5)}$$

Menghitung kembali sesuai dengan fungsi aktifasi

$$\boldsymbol{y_k = f(y_{in_k})} \quad \textbf{(2-6)}$$

3. Propagasi Mundur *(Backpropagation)* dan Galatnya

Langkah 5 :

Masing-masing *unit* keluaran (Yk, k=1,…,m) menerima pola target sesuai dengan pola masukan saat pelatihan dan dihitung galatnya :

$$\boldsymbol{\delta_k = (t_k - y_k) f'(y_in_k)} \quad \textbf{(2-7)}$$

Karena $f'(y_in_k) = y_k$ menggunakan fungsi sigmoid, maka :

$$\boldsymbol{f'(y_in_k) = f(y_in_k)(1 - f(y_in_k))}$$
$$\boldsymbol{= y_k(1 - y_k)} \quad \textbf{(2-8)}$$

Menghitung perbaikan faktor penimbang (kemudian untuk memperbaiki $\boldsymbol{w_{jk}}$

$$\boldsymbol{\Delta W_{kj} = \alpha . \partial_k . Z_j} \quad \textbf{(2-9)}$$

Menghitung perbaikan koreksi :

$$\boldsymbol{\Delta W_{0j} = \alpha . \partial_k} \quad \textbf{(2-10)}$$

Dan menggunakan nilai δ_k pada semua *unit* lapisan sebelumnya.

Langkah 6:

Masing-masing penimbang yang menghubungkan *unit-unit* l;apisan keluaran dengan *unit-unit* pada lapisan tersembunyi $Z_j, j = 1, \dots, p$) dikalikan delta dan dijumlahkan sebagai masukan ke *unit-unit* lapisan berikutnya.

$$\boldsymbol{\delta_in_j = \sum_{k=1}^{n} \delta_k W_{jk}} \quad \textbf{(2-11)}$$

Selanjutnya dikalikan dengan turunan dari fungsi aktifasinya untuk menghitung galat.

$$\boldsymbol{\delta_j = \delta_in_j\ f'(y\ _in_j)} \quad \textbf{(2-12)}$$

Kemudian menghitung perbaikan penimbang (digunakan untuk memperbaiki **V_ij)**

$$\boldsymbol{\Delta V_ij = \alpha \delta_j\ X_i} \quad \textbf{(2-13)}$$

Kemudian menghitung perbaikan bias (untuk memperbaiki **V_oj**)

$$\boldsymbol{\Delta V_oj = \alpha \delta_j} \quad \textbf{(2-14)}$$

Memperbaiki penimbang dan bias

Langkah 7 :

Masing-masing keluaran *unit* (yk, k=1,…,m) diperbaiki bias dan penimbangnya (j=0,…,p)

$$\textbf{W_jk (baru)=W_jk (lama)}\Delta\textbf{W_jk} \quad \textbf{(2-15)}$$

Masing-masing *unit* tersembunyi (Zj, j:1,…,p) diperbaiki bias dan penimbangnya (j=0,…,n)

$$\textbf{V_jk (baru)=V_jk (lama)}\Delta\textbf{V_jk} \quad \textbf{(2-16)}$$

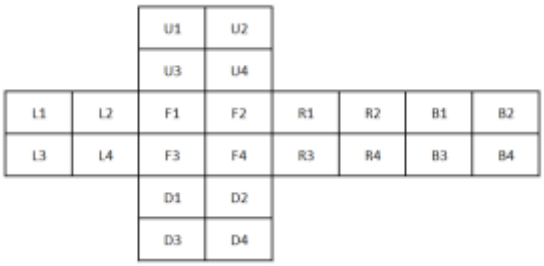
Langkah 8 :

Uji kondisi pemberhentian (akhir iterasi) pengujian pemberhentian dilakukan dengan menghitung nilai mean *Square Error* (*MSE*). Di mana nilai *MSE* ini akan mengambil rata-rata kuadrat *error* yang terjadi antara *output* jaringan dan target, berikut adalah rumus *MSE*

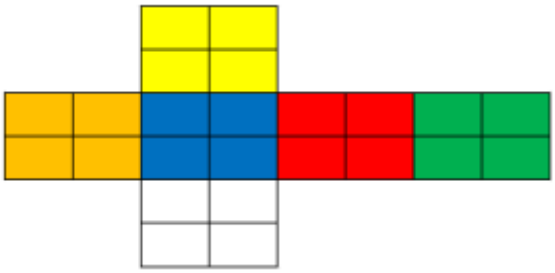
$$\textbf{MSE} = \frac{\sum_{i=1}^{n} e_j^2}{n} \quad \textbf{(2-17)}$$

2. ISI PENELITIAN

Rubik's CUBE 2×2 sesuai dengan konfigurasi yang telah disediakan oleh aplikasi. Dalam kasus ini banyaknya baris dan kolom pada *Rubik's* adalah 2×2. Analisis data masukan dilakukan untuk mendapatkan sebuah nilai *input* an yang nantinya dapat digunakan untuk proses mengklasifikasikan dalam metode *Backpropagation*. *Input* an tersebut berupa matriks *Data Set* yang didapat dari aplikasi yang telah ditentukan, merupakan hasil ekstraksi ciri dari setiap langkah. Dibawah ini adalah tampilan dari setiap *Rubik's* 2x2.



Gambar 3.Tampilan *Rubik's* 1



Gambar 4.Tampilan *Rubik's* 2

Tabel 1.Konversi Warna *Rubik's*

No	Warna	Keterangan
1	*RED*	R
2	*GREEN*	G
3	*BLUE*	B
4	*YELLOW*	Y
5	*WHITE*	W
6	*ORANGE*	O

2.1 Konversi Kode Notasi *Rubik's*

Terminologi yang biasa dipakai untuk menjelaskan suatu Algoritma *Rubik's CUBE* sebagai berikut:

1. *Rubik's CUBE* terdiri atas enam buah sisi: depan (*front*), belakang (*back*), kiri (*left*), kanan (*right*), atas (*up*), dan bawah (*down*).
2. Gerakan memutar sisi *Rubik's CUBE* searah jarum jam disimbolkan dengan huruf pertama dari nama sisi tersebut dalam bahasa Inggris. Untuk yang berlawanan arah dengan jarum jam, ditambahkan tanda petik (’). Contohnya, F berarti memutar sisi depan (*front*) searah jarum jam, sedangkan L’ berarti memutar sisi kiri (*left*) berlawanan arah dengan jarum jam.

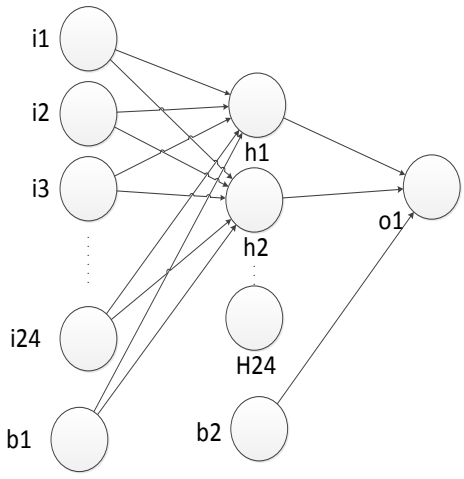
Pada aplikasi yang akan dirancang kode notasi tersebut mengalami perubahan pada Algoritma yang digunakan dalam aplikasi. Perubahan ini bertujuan agar sistem lebih mudah dalam membaca perintah untuk menjalankan kode notasi yang dituliskan. Perubahan dapat dilihat pada tabel sebagai berikut :

Tabel 2.Konversi Notasi Rubik's

No	Keterangan		Simbol
1	*UP*	Memutar ke atas searah jarum jam	U
2	*UP'*	Memutar ke atas berlawanan jarum jam	U'
3	*Down*	Memutar ke bawah searah jarum jam	D
4	*Down'*	Memutar ke bawah berlawanan jarum jam	D'
5	*FRONT*	Memutar ke depan searah jarum jam	F
6	*FRONT'*	Memutar ke depan berlawan jarum jam	F'
7	*BACK*	Memutar ke belakang searah jarum jam	B
8	*BACK'*	Memutar ke belakang berlawanan jarum jam	B'
9	*LEFT*	Memutar ke kiri searah jarum jam	L
10	*LEFT'*	Memutar ke kiri berlawanan jarum jam	L'
11	*RIGHT*	Memutar ke kanan searah jarum jam	R
12	*RIGHT'*	Memutar ke kanan berlawanan jarum jam	R'

2.2 Analisis Pelatihan Metode Backpropagation

Langkah-langkah dengan menggunakan Algoritma *Backpropagation* akan dijelaskan dalam subbab analisis metode. Nilai ini akan menjadi acuan dalam melakukan *Testing*. Dalam mencari fitur setiap kelas maka perlu adanya pelatihan. Untuk lebih jelasnya, arsitektur analisis dapat dilihat pada gambar sebagai berikut.



Gambar 5.Arsitektur Jaringan

Parameter arsitektur jaringan untuk menghitung *Backpropagation* adalah sebagai berikut :

1. i = contoh *input*an nilai 24 ciri, didapatkan dari jumlah semua kotak yang terdapat pada rubik sebanyak 24.
2. h = contoh *Hidden Layer* 36. *Hidden Layer* merupakan layer tersebunyi yang digunakan sebagai pembanding. Karena itu fungsi dari *Hidden Layer* adalah menambahkan instruksi logika, jadi semakin banyak logika yang ditambahkan atau semakin banyak *Hidden Layer* maka memory yang digunakan semakin banyak dan akan menghasilkan proses yang semakin lama.
3. b = bias. Sementara bias digunakan untuk masukan dengan nilai tetap sama yaitu 1, penggunaan bias untuk mempercepat pelatihan untuk mengoreksi terhadap kecukupan variabel-variabel *input* yang ditetapkan.
4. O = *output*. Pada nilai *output* terdapat nilai untuk menentukan pergeseran karakter sampai mencapai *solve*.

Kemudian mencari nilai normalisasi pergeseran, nilai normalisasi dimaksudkan untuk mencari *threshold* yang digunakan sebagai ketentuan dalam proses *Backpropagation*. *Threshold* sendiri adalah jarak antar nilai normalisasi, sedangkan normalisasi adalah nilai penentu tepi *threshold.* Nilai normalisasi yang didapatkan menggunakan persamaan sebagai berikut :

$$V' \frac{\boldsymbol{nilai\ karakter - nilai\ karakter\ terkecil}}{\boldsymbol{nilai\ karakter\ terbesar - nilai\ karakter\ terkecil}}$$

Sebagai contoh menggunakan contoh pada pergeseran F (Front), didapatkan proses dan hasil seperti berikut :

$$\boldsymbol{V' = \frac{1-0}{12-0}}$$
$$\boldsymbol{V' = 0,083}$$

Setelah dihitung semua nilai sampai data ke 12, maka nilai yang ada dapat dilihat pada tabel berikut :

Tabel 3.Pergerakan Karakter

Pergeseran	Nilai normalisasi pergeseran	Treshold
F (Front)	0,083	0 - 0.083
B (Back)	0,16	0.084 - 0.16
D (Down)	0,25	0.17 - 0.25
U (Up)	0,33	0.26 - 0.33
R (Right)	0,42	0.34 – 0.42
L (Left)	0,50	0.43 - 0.50
F’	0,58	0.51 – 0.58
B’	0,67	0.59 – 0.67
D’	0,75	0.68 – 0.75
U’	0,83	0.76 – 0.83
R’	0,92	0.84 – 0.92
L’	1	0.93 - 1

Pada kasus ini nilai yang digunakan untuk *Learning Rate* adalah 0,5. Karena Algoritma *Backpropataion* hanya dapat menerima *input*an dari 0 – 1 maka nilai *input* dan *output* yang tidak sesuai dengan aturan *Backpropagation.* Sehingga bisa dituliskan nilai normalisasi sebagai berikut :

Tabel 4.Normalisasi Nilai

No	Nilai
0	0
1	0,083
2	0,16
3	0,25
4	0,33
5	0,42
6	0,50
7	0,58
8	0,67
9	0,75
10	0,83
11	0,92
12	1

Dari nilai yang sudah dibentuk di atas kemudian digunakan untuk diperhitungan Algoritma *Backpropagation.* Langka-langkah perhitungannya adalah sebagai berikut. Algoritma *Neural Network Backpropagation* ini adalah jenis Algoritma *supervised learning*, sehingga Algoritma *Backpropagation* akan optimal ketika memiliki banyak data *Training*. Dari *Data Set* yang sudah dibentuk diatas kemudian digunakan untuk *input*an diperhitungan Algoritma *Backpropagation*. Langka-langkah perhitungannya adalah sebagai berikut.

1. Data masukan

   Untuk membuat mewakili *Rubik’s* dalam menghitung penyelesaiannya, yang perlu diperhatikan adalah stiker pembentuk *Rubik’s* yang masih dalam keadaan acak yaitu bagian bawah, belakang, kiri kubus. Itu artinya *Rubik’s* dengan stiker putih, merah dan biru / warna pada data masukan tidak bergerak ketika

memanipulasi kubus karena sudah dilakukan skala saat melakukan perhitungan. Sebuah *Rubik's* dilakukan perhitungan ke program dengan memberikan urutan 24 ruang dipisahkan karakter yang sesuai dengan warna.

Karakter warna:
y: kuning
g: hijau
w: putih
b: biru
r: merah
o: oranye

Stiker yang disebutkan (1-9 maka A-Q) seperti menggunakan ini membuka peta 2-d kubus sebagai berikut :

Tabel 5.Data Masukan *Rubik's*

		1	**2**		
		3	**4**		
H	**I**	**5**	**6**	**N**	**O**
J	**K**	**7**	**8**	**P**	**Q**
		9	**A**		
		B	**C**		
		D	**E**		
		F	**G**		

Jadi, setiap warna stiker dapat dikodekan seperti berikut :
putih = 000
kuning = 001
biru = 010
hijau = 011
merah = 100
oranye = 101

Kemudian dilihat dari 6 sisi *Rubik's*, setiap sisi dapat berubah dengan memberikan total 12 kemungkinan dan dapat dikodekan ke dalam keluaran dengan cara yang sama, seperti contoh empat pergerakan berikut :
R = 0000 R '= 1000 L = 0001 L' = 1001

Setelah menentukan gerakan yang dimaksud, dapat dilakukan perhitungan nilai *Hidden* dan *Net* dimana *input*an harus dinormalisasi dulu dengan menggunakan rumus sigmoid bipolar menggunakan persamaan (2-1) sebagai berikut :

Tabel 6.Nilai Gerakan *Rubik's*

Nama Gerakan	Bentuk Biner	Bentuk Nilai
R	0000	0
R'	1000	8
L	0001	1
.....	.....	.....
L'	1001	9

$$y = f(x) = \frac{1 - e^{-x}}{(1 + e^{-x})}$$

W1 $= f(R) = \frac{1-e^{-0}}{(1+e^{-0})} =$ **0,732756**

W2 $= f(R) = \frac{1-e^{-8}}{(1+e^{-8})} =$ **0,126772**

W3 $= f(R) = \frac{1-e^{-1}}{(1+e^{-1})} =$ **0,754919**

Dan selanjutnya sampai W12,

W12 $= f(R) = \frac{1-e^{-9}}{(1+e^{-9})} =$ **0,002581**

1. Menentukan Nilai Bobot

Proses pertama ini juga bertujuan untuk mengoptimalkan pemilihan jalur penggabungan karakter ketika data *Training* yang digunakan masih sedikit. Hitung nilai bobot, yaitu parameter w dan v sebagai bobot, hasil ini didapatkan dari hasil dari pembelajaran data set secara random yang sudah ada. Misal dari data yang sudah masuk sebagai berikut :

Tabel 7.Nilai Bobot W

w1	0,732756
w2	0,126772
w3	0,754919
w4	0,111275
w5	0,499606
w6	0,294177
w7	0,556117
w8	0,884137
w9	0,357525
w10	0,851517
w11	0,162618
W12	0,002581

Setelah mendapatkan hasil bobot W dari *Data Set* acak, Kemudian menentukan bobot V yang didapatkan ketika melakukan diidentifikasi sebagai *Hidden Layer* dengan nilai sebagai berikut :

Tabel 8.Hidden Layer Bobot V

v1	**0,489456**
v2	**0,616634**
.	.
.	.
.	.
V12	**0,840474**

2. Propagasi maju (*Feedforward*)

a. Hitung Nilai *Net*

Untuk menghitung nilai net, *input*an harus dinormalisasi dulu dengan menggunakan rumus sigmoid bipolar, harus dinormalisasi dengan menggunakan persamaan **(2-1)**, Dimana parameter yang akan digunakan adalah :
Learning Rate = 0,5
Target *error* = 0,1
Sebelum mencari nilai net, menentukan secara acak nilai aktifasi *unit* yang jumlahnya sama dengan nilai W yaitu sebanyak 162. Namun

dalam contoh perhitungan akan digunakan 5 nilai sebagai berikut :

Tabel 9.Aktivasi Unit 1

Jenis	I1	I2	I3	I4	I5
nilai	1	0,646082206	0,956740301	0,516962148	0,995244031

Setelah mendapatkan nilai I, maka langkah selanjutnya adalah mencari nilai net, persamaan yang akan digunakan adalah persamaan **(2-2)**. Dengan kasus analisis ini dijelaskan bahwa $X_i = W_i$ dan dan $V_{ij} = I_{ij}$

b. Aktivasi Nilai Net
Untuk mengaktivasi nilai net yang sudah dihasilkan dari perhitungan diatas maka di dapatkan nilai hasil h1 dengan nilai 2,091677417

c. Hitung *Output*
Nilai *output* disini adalah nilai dari klasifikasi, persamaan yang akan digunakan untuk mendapatkan nilai adalah persamaan **(2-2)** sehingga dapat dituliskan dengan :
Output = v1 * h1
Output = 0,489456 * 2,091677417
Output = 1,023784061

d. Nilai Target Klasifikasi
Nilai target klasifikasi digunakan untuk menentukan klasifikasi dari *input*an yang akan di klasifikasikan. Dengan klasifikasi pada tabel berikut :

Tabel 10.Nilai Target Klasifikasi

Pergeseran	Treshold
F (Front)	0 - 0.083
B (Back)	0.084 - 0.09
D (Down)	0.10 - 0.1
U (Up)	0.12 - 0.111
R (Right)	0.112 – 0.125
L (Left)	0.126 - 0.143
F’	0.144 – 0.167
B’	0.168 – 0.2
D’	0.21 – 0.25
U’	0.26 – 0.33
R’	0.34 – 0.5
L’	0.6 - 1

e. Hitung Nilai *Error*
Nilai *error* untuk menentukan klasifikasi, jika nilai sudah memenuhi kriteria target klasifikasi maka akan dilakukan pergeseran, sebaliknya maka dilakukan proses *Backpropagation* berikutnya. Karena didapat dari nilai *output* 1,023784061, sehingga diketahui melebihi target klasifikasi, maka dilakukan proses *Backpropagation* selanjutnya.

3. Propagasi Mundur (*Backpropagation*) dan Galatnya
Metode *Backpropagation* akan diterapkan pada analisis masalah pada pencarian untuk menyusun warna yang terac ak pada permainan *Rubik’s CUBE*, proses ini dilakukan untuk perubahan bobot sehingga mendapatkan nilai *error* yang mendekati target *Error*. Tahapan yang akan di lakukan.:

a. Mencari nilai delta 1
Delta 1 digunakan untuk perhitungan nilai delta v
Perhitungan yang akan dilakukan
Delta 1 =(target – *output*1)**output*1*(1-*output*1)
Delta 1 =(0 - 1,023784061) * 1,023784061* (1-1,023784061)
Delta 1 = 0,024928878
Kemudian menentukan delta W dengan melatih *output* tersebut dengan langkah yang tersembunyi yang dilakukan.

b. Mencari nilai delta w
Delta w digunakan untuk mendapatkan nilai delta v. Dengan perhitungan di bawah ini :
Delta w = *Learning Rate* * delta 1 * *output* hidden i
Delta w = 0,5 * 0,024928878 * 1,023784061
Delta w = 0,012760893

c. Mencari nilai delta net
Delta net digunakan untuk mendapatkan perhitungan pada pencarian nilai *output*. Dengan perhitungan berikut :
Delta net = delta w * nilai random v
Delta net = 0,012760893 * 0,489456
Delta net = 0,006245895

d. Mencari nilai delta *output*
Delta *output* atau i digunakan untuk mendapatkan perhitungna dalam mencari nilai bobot w. Perhitungan yang akan dilakukan sebagai berikut :
Delta i = delta net **output* hidden i *(1-*output* hidden i)
Delta i = 0,006245895 * 1,023784061 * (1-1,023784061)
Delta i = -0,000152085

e. Mencari nilai bobot w
Bobot w digunakan untuk mendapatkan nilai bobot baru atau update bobot. Perhitungan sebagai berikut :
Bobot w = *Learning Rate* * delta i * *input* i
Bobot w = 0,5 * (-0,000152085) * 1
Bobot w = -0,000076042

f. Update bobot
Update bobo digunakan untuk menetukan nilai dari bobo baru.
Update bobot = bobot lama + bobot wi
Update bobot = 0,732756 + (-0,000076042)
Update bobot = 0,732679958

Dari hasil update bobot yang sudah dilakukan untuk memperoleh nilai *error* yang lebih kecil dari hasil langkah sebelumnya. Dan dengan nilai 0,732679958 sehingga memenuhi target kriteria D'.
Begitu seterusnya iterasi pergerakan dilakukan berulang kali sampai menghasilkan *solve*. Dibawah ini adalah hasil ahir dari setiap perhitung perputaran rubik :

Tabel 11.Hasil Perhitungan

No	Nilai Hasil Bobot	Notasi Gerakan	Keterangan
1	0,934345	L'	Memutar ke kiri berlawanan jarum jam
2	0,656757	B'	Memutar ke belakang berlawanan jarum jam
3	0,773435	U'	Memutar ke atas berlawanan jarum jam
4	0,854656	R'	Memutar ke kanan berlawanan jarum jam
5	0,548986	F'	Memutar ke depan berlawanan jarum jam

3. PENUTUP

Berdasarkan hasil yang didapat dalam penelitian dan penyusunan skripsi ini serta disesuaikan dengan tujuan penelitian maka diperoleh kesimpulan sebagai berikut:

1. Aplikasi sudah dapat menentukan tahapan penyelesaian penelitian menggunakan Algoritma *Backpropagation.*
2. Akurasi waktu dari pengujian Algoritma *Backpropagation* yang paling cepat terselesaikan yaitu dengan rata-rata waktu 0,3275 detik sedangkan hasil pengujian akurasi waktu yang paling lama terselesaikan yaitu dengan rata-rata waktu 2,321 detik.

DAFTAR PUSTAKA

[1]. Rubik's Official Website, "Rubik's World Sube Fact", http://www.Rubik's.com/world/ CUBE_facts.php (Diakses pada tanggal 22 September 2015).

[2]. Richard E. Korf , Larry A. Taylor, "Pruning Duplicate Nodes in Depth-First Search", Proceedings of the Eleventh National Conference on Artificial Intelligence, Washington, D.C, pp. 756-761, July 1993.

[3]. Tomas Rokicki Herbert Kociemba, "God's Number is 20", http://www.CUBE20.org (Diakses pada tanggal 22 September 2015).