ANALISIS VARIASI PARAMETER *LEARNING VECTOR QUANTIZATION ARTIFICIAL NEURAL NETWORK* TERHADAP PENGENALAN POLA DATA *ODOR*

PARAMETER VARIATION ANALYSIS OF LEARNING VECTOR QUANTIZATION ARTIFICIAL NEURAL NETWORK FOR ODOR DATA PATTERN RECOGNITION

Ikhwannul Kholis

Universitas 17 Agustus 1945 – Jakarta
ikhwanul.kholis@uta45jakarta.ac.id

Abstrak

Pengenalan pola data *odor* dapat dilakukan dengan menggunakan metode *Learning Vector Quantization* (LVQ) *Artificial Neural Network* (ANN). ANN dibuat menyerupai sistem syaraf manusia, disebut juga Jaringan Syaraf Tiruan (JST). Dengan beberapa parameter pada LVQ, dapat diketahui karakteristik LVQ sehingga dapat memperkecil *error* dan *epoch*, serta memperbesar *Recognition Rate*. Hasil percobaan menunjukkan hubungan antara parameter *alpha*, konstanta laju pembelajaran, jumlah *epoch*, dan inisialisasi *vector* pewakil terhadap *error* dan *Recognition Rate* yang diperoleh.

Kata Kunci: ANN, *Learning Vector Quantization, epoch, error,* JST, *Recognition Rate*.

Abstract

Pattern recognition of odor data can be done by using Learning Vector Quantization Artificial Neural Network (ANN). ANN is made to resemble the human neural system. By varying some parameters on Learning Vector Quantization, Learning Vector Quantization characteristics can be identified to minimize errors and epoch and to enlarge Recognition Rate. The experimental results showed the relationship between the alpha parameter, coefficient alpha, the number of epoch, and vector initialization against error and the Recognition Rate obtained.

Keywords: *ANN, Learning Vector Quantization, epoch, error, JST, Recognition Rate.*

Tanggal Terima Naskah : 22 April 2015
Tanggal Persetujuan Naskah : 10 Agustus 2015

1 PENDAHULUAN

Salah satu cabang dari *Artificial Intelligence* (AI) adalah apa yang dikenal dengan Jaringan Saraf Tiruan (*Artificial Neural Network*). Jaringan syaraf tiruan telah dikembangkan sejak tahun 1940. Pada tahun 1943, McCulloch dan W.H.Pitts memperkenalkan pemodelan matematis neuron [1]. Tahun 1949, Hebb mencoba mengkaji proses belajar yang dilakukan oleh neuron. Teori ini dikenal sebagai Hebbian Law. Tahun 1958, Rosenblatt memperkenalkan konsep perseptron suatu jaringan yang

terdiri dari beberapa lapisan yang saling berhubungan melalui umpan maju (*feed foward*). Konsep ini dimaksudkan untuk memberikan ilustrasi tentang dasar-dasar intelegensia secara umum. Hasil kerja Rosenblatt yang sangat penting adalah *perceptron convergence theorem* (tahun 1962) yang membuktikan bahwa bila setiap perseptron dapat memilah-milah dua buah pola yang berbeda maka siklus pelatihannya dapat dilakukan dalam jumlah yang terbatas.

Sensor penciuman manusia dapat mengenali berbagai aroma. Sistem pengenalan aroma tersebut terletak pada indera penciuman manusia. Pengenalan data *odor* pada suatu sistem sering dilakukan untuk berbagai tujuan, sepert pendeteksian gas, pendeteksian *air quality*, dan sebagainya. Dengan demikian, diperlukan sistem dengan pengenalan data *odor* yang baik.

Pengenalan pola data *odor* dapat dilakukan dengan menggunakan ANN *Learning* Vektor *Quantization*. Data set dari aroma dapat diklasifikasikan terhadap jenis dari *odor* tertentu. Dengan demikian, klasifikasi aroma dapat dilakukan lebih mudah dengan menggunakan metode ANN *Learning* Vektor *Quantization* (LVQ).

2 KONSEP DASAR

2.1 Pengertian *Learning* Vektor *Quantization*

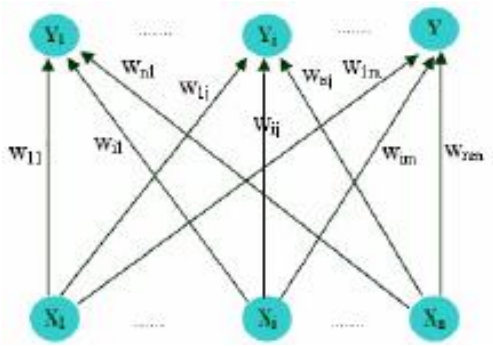
Jaringan Syaraf Tiruan (JST) merupakan salah satu sistem pemrosesan informasi atau data yang didesain dengan meniru cara kerja otak manusia dalam menyelesaikan suatu masalah dengan melakukan proses belajar melalui perubahan bobot sinapsisnya. Salah satu metode yang digunakan dalam JST adalah *Learning* Vektor *Quantization* (LVQ). Kuantisasi vektor pembelajaran atau *Learning Vektor Quantiation* (LVQ) merupakan metode pengklarifikasi pola dengan setiap unit keluaran mewakili satu kelas tertentu atau satu kategori tertentu [2]. Beberapa unit keluaran harus digunakan untuk masing-masing kelas. Vektor bobot dari satu unit keluaran sering disebut sebagai vektor acuan bagi kelas/kategori yang diwakili oleh keluaran tersebut.

Dalam proses pelatihannya, unit-unit keluaran ini digerakkan dan diarahkan (melalui pembaharuan nilai bobot-bobot dalam pelatihan dengan pengarahan) untuk mendekati suatu permukaan keputusan (*decision surface*) dalam teori pengklasifikasi Bayes. Hal ini dilakukan dengan asumsi bahwa sekumpulan pola-pola pelatihan dengan masing-masing klasifikasi yang berkaitan telah disediakan, berikut juga distribusi awal dari vektor-vektor acuan yang mewakili setiap klarifikasi tersebut.

Setelah melakukan proses pelatihan, jaringan LVQ ini dapat mengklarifikasi suatu vektor masukan, dengan menempatkannya pada kelas yang sama dengan unit keluaran yang mempunyai vektor bobot (dalam hal ini merupakan vektor acuan) paling sesuai dengan vektor masukan tersebut.

2.2 Arsitektur *Learning* Vektor *Quantization*

Arsitektur algoritma *Learning Vector Quantization* terdiri atas dua *layer*, yaitu *input layer*, dan *output layer*. Pada *input layer* tidak terjadi proses komputasi, hanya terjadi pengiriman sinyal *input* ke *output layer*. Pada *output layer* terjadi proses komputasi terhadap vektor bobot dengan menggunakan laju pembelajaran [3]. *Update* laju pembelajaran juga dilakukan dengan menggunakan koefisien laju pembelajaran.



Gambar 1. Arsitektur ANN *learning vector quantization*

dimana:
W_{ij} = Bobot pada lapisan keluaran (*output layer*)
X = Lapisan masukan (*Input Layer*)
Y = Lapisan keluaran (*Output Layer*)

2.2 Algoritma *Learning Vector Quantization*

2.2.1 Inisialisasi Laju Pembelajaran *Nguyen-Widrow*

Terdapat dua cara untuk menginisialisasi bobot, yaitu inisialisasi secara *random* dan inisialisasi dengan menggunakan *Nguyen-Widrow*. Inisialisasi acak merupakan cara yang paling sering digunakan dalam inisialisasi bobot. Pada inisialisasi bobot secara *random*, bobot diinisialisasi secara acak tanpa menggunakan faktor skala. Pada inisialisasi *Nguyen-Widrow*, inisialisasi dilakukan dengan memodifikasi inisialisasi acak dengan menggunakan faktor skala β dengan tujuan untuk mempercepat proses pelatihan.

Algoritma inisialisasi dengan *Nguyen-Widrow* adalah sebagai berikut:

a. Menentukan besarnya skala β

$$\beta = 0.7(\mathrm{p})^{1/n} \quad \text{.....(1)}$$

dengan p = jumlah unit *hidden* dan n= jumlah unit *input*.

b. Inisialisasi bobot W_{ij} secara *random* dengan nilai inisialisasi W_{ij} adalah

$$-0.5 \le W_{ij} \le 0.5 \quad \text{.....(2)}$$

c. Menghitung besarnya *magnitude* bobot W_{ij}

$$\|W_{ij}\| = \sqrt{\left\{\sum_{i=1}^{p} (W_{ij})^2\right\}} \quad \text{.....(3)}$$

d. Meng-*update* bobot W_{ij}

$$W_{ij} = \frac{\beta . W_{ij}}{\|W_j\|} \quad \text{.....(4)}$$

2.2.2 Inisialisasi Vektor Acuan/Pewakil

Dalam LVQ, digunakan Vektor Pewakil sebagai vektor acuan bagi masukan untuk mendekati nilai vektor pewakil tersebut. Vektor Pewakil (VP) diperoleh dari proses inisialisasi Vektor Pewakil. Beberapa metode untuk melakukan inisialisasi vektor pewakil diberikan sebagai berikut:

1. Inisialisasi Vektor Pewakil dengan data dari tiap kelas
2. Inisialisasi Vektor Pewakil dengan angka *random*
3. Inisialisasi Vektor Pewakil dengan memilih secara *random* dari masukan
4. Inisialisasi Vektor Pewakil dengan angka nol

2.2.3 *Learning Vector Quantization*

Tujuan dari algoritma jaringan LVQ adalah mendapatkan unit keluaran yang paling mirip dengan vektor masukan sehingga pada akhirnya akan diperoleh kondisi: apabila x dan wc merupakan bagian dari kelas yang sama, maka vektor bobot akan digeser mendekati vektor masukan ini; dan apabila x dan wc merupakan bagian dari kelas yang berbeda, vektor bobot akan digeser untuk menjauhi vektor masukan tersebut.
Penamaan yang digunakan adalah sebagai berikut:

1. x merupakan vektor pelatihan (x1, ..., xi, ...,xn)
2. T merupakan kategori atau kelas yang benar untuk vektor pelatihan.
3. Wj merupakan vektor bobot untuk unit keluaran ke-j (w1j, . .,wij, . . .,win)
4. Cj merupakan kategori atau kelas yang diwakili oleh unit keluaran ke-j.
5. ||x-wj|| merupakan jarak *Euclidean* terkecil antara vektor masukan dan vektor bobot dari unit keluaran ke-j.

Langkah 1: Inisialisasi vektor-vektor acuan dan inisialisasi laju pembelajaran $\alpha(0)$.
Langkah 2: selama syarat henti tidak terpenuhi, lakukan langkah 3-7
Langkah 3 : untuk setiap vektor masukan (pelatihan) x, lakukan langkah 4 – 5
Langkah 4 : cari J sehingga ||x-wj|| berharga minimum
Langkah 5 : perbaharui wj dengan menggunakan persamaan berikut,

$$W_j(n) = W_j(n-1) + \alpha[x - W_j(n-1)]; \text{ dengan } T = C_j$$
$$W_j(n) = W_j(n-1) - \alpha[x - W_j(n-1)]; \text{ dengan } T \neq C_j \quad \text{.....(5)}$$

Langkah 6 : kurangi harga laju pembelajaran
Langkah 7 : uji syarat henti.

2.2.4 Parameter Pelatihan

2.2.4.1 Laju Pembelajaran (α)

Laju pembelajaran merupakan parameter jaringan dalam mengendalikan proses penyesuaian bobot. Nilai laju pembelajaran yang optimal bergantung pada kasus yang dihadapi. Laju pembelajaran yang terlalu kecil menyebabkan konvergensi jaringan menjadi lebih lambat, sedangkan laju pembelajaran yang terlalu besar dapat menyebabkan ketidakstabilan pada jaringan.

2.2.4.2 Koefisien Laju Pembelajaran

Koefisien Laju Pembelajaran berfungsi untuk mengkonvergenkan nilai perubahan bobot. Nilai dari Koefisien Laju Pembelajaran ini berkisar antara 0 < koefisien laju pembelajaran < 1.

2.2.4.3 Metode Pembobotan Awal

Percobaan dilakukan dengan beberapa cara dalam menentukan bobot awal (vektor pewakil awal), yaitu:

1. Inisialisasi vektor pewakil dengan data dari tiap kelas untuk masing-masing vektor pewakil dari data yang telah diberikan
2. Inisialisasi dengan angka *random*
3. Inisialisasi dengan *random*, tetapi dari data yang telah diberikan
4. Inisialisasi vektor pewakil awal dengan nilai nol

2.2.5 *Stopping Condition*

Terdapat kondisi henti (*stopping condition*) pada algoritma *Learning* Vektor *Quantization* ini, yaitu *Epoch* > *Epoch* maksimum. *Epoch* adalah suatu langkah yang dilakukan dalam pembelajaran pada ANN. Jika besarnya *epoch* lebih besar dari besarnya *epoch* maksimum yang telah ditetapkan, maka proses pembelajaran akan berhenti. Dengan demikian, kondisi henti terjadi jika besarnya *Epoch* > *Epoch* maksimum.

2.3 *Principal Component Analysis*

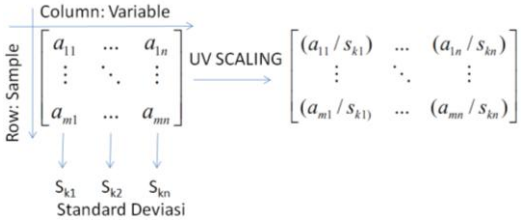
Principal Component Analysis (PCA) merupakan suatu teknik untuk menyederhanakan suatu *dataset*, dengan mengurangi *dataset* multidimensional ke dimensi yang lebih rendah dengan cara mengambil bagian-bagian dimensi yang penting (*orthogonal* secara vektor). Tujuan dari PCA adalah mengurangi dimensionalitas data sambil tetap memelihara sebanyak mungkin variansi yang muncul dalam *dataset*.

Berikut adalah langkah-langkah dalam melakukan PCA:

a. *Scaling*

Proses *Scaling* bertujuan untuk membuat suatu *dataset* memiliki persebaran data yang lebih baik. *Output* dari proses *scaling* adalah suatu *Matrix Auto Scaling*, yang merupakan pengurangan dari *Matrix* UV *Scaling* dengan *Matrix Mean Centering*.

Matrix UV *Scaling* adalah *matrix dataset* yang telah dibagi dengan standar deviasi dari setiap dimensi (dalam hal ini dimensi diwakili oleh kolom). Berikut adalah ilustrasi *matrix* UV *Scaling*:



Gambar 2. *Scaling matrix*

Proses *mean centering* adalah proses pencarian nilai rata-rata setiap dimensi (dalam program yang dibuat diwakili dengan nama *variable*). Setelah melakukan proses *mean centering*, *matrix* UV *scaling* dikurangkan dengan nilai rata-rata sehingga terbentuk *matrix autoscaling*. Berikut adalah potongan program pca.m yang telah dibuat, yang terdapat potongan proses *scaling*.

```
function [SD,X_ave,Xi_auto,T] = pca(Xi,L)
%Xi          matrix input
%L           dimensi akhir
%SD          Standar Deviasi
%X_ave       Rata-rata X
%Xi_auto     Auto Scaling Marix

m = size(Xi,1);      %m x n matrix, dapet m (baris)
n = size(Xi,2);      %m x n matrix, dapet n (kolom)

%Scaling
SD = std(Xi);        %standar deviasi

%UV Scaling process
for sampel_ke = 1:n
    Xi_scaled(:,sampel_ke) = Xi(:,sampel_ke)/SD(sampel_ke);
end

%Mean Centering
X_ave = mean(Xi_scaled);

%Matrix auto
for sampel_ke = 1:n
    Xi_auto(:,sampel_ke) = Xi_scaled(:,sampel_ke)-X_ave(sampel_ke);
end
```

Gambar 3. Program proses *scaling*

Pada program pca.m yang telah dibuat, matrix Xi_scaled merupakan *matrix autoscaling*, X_ave merupakan nilai rata-rata untuk proses *mean centering*, dan Xi_auto adalah *matrix auto-scaling*.

b. Menghitung *Covariance Matrix*
Covariance matrix dihitung dengan persamaan:

$$S = \left(\frac{1}{m-1}\right) X^T X = \mathrm{cov}(X) \quad \text{.....(6)}$$

Pada MATLAB, nilai *covariance matrix* dapat dicari dengan menggunakan fungsi S=cov(X), dengan X adalah *matrix input* dan S adalah *covariance matrix*.

c. Menghitung Nilai *Eigen* dan Vektor *Eigen*
Pencarian Nilai *eigen* dan vektor *eigen* bertujuan untuk mengetahui sejauh mana dimensionalitas *dataset* dapat dipotong, dimensi yang memiliki *eigen value* yang besar adalah dimensi yang dinilai sangat penting dalam *dataset*. Pencarian *eigen value* dalam MATLAB dapat dilakukan dengan fungsi [P,A]=eig(S), dengan S adalah *matrix input*, P merupakan vektor *eigen* dan A merupakan nilai *eigen*.

d. Menghitung *Score*
Score merupakan nilai akhir *dataset* yang dapat mewakili *dataset*. *Score* dihitung dengan persamaan:

$$[X]([P]^T)^{-1} = [T] \quad \text{.....(7)}$$

dimana X adalah *matrix autoscaling*, P merupakan *eigen vector* dan T merupakan *score* dari proses PCA. Berikut merupakan potongan program dari proses perhitungan *covariance matrix* sampai mendapatkan *score*:

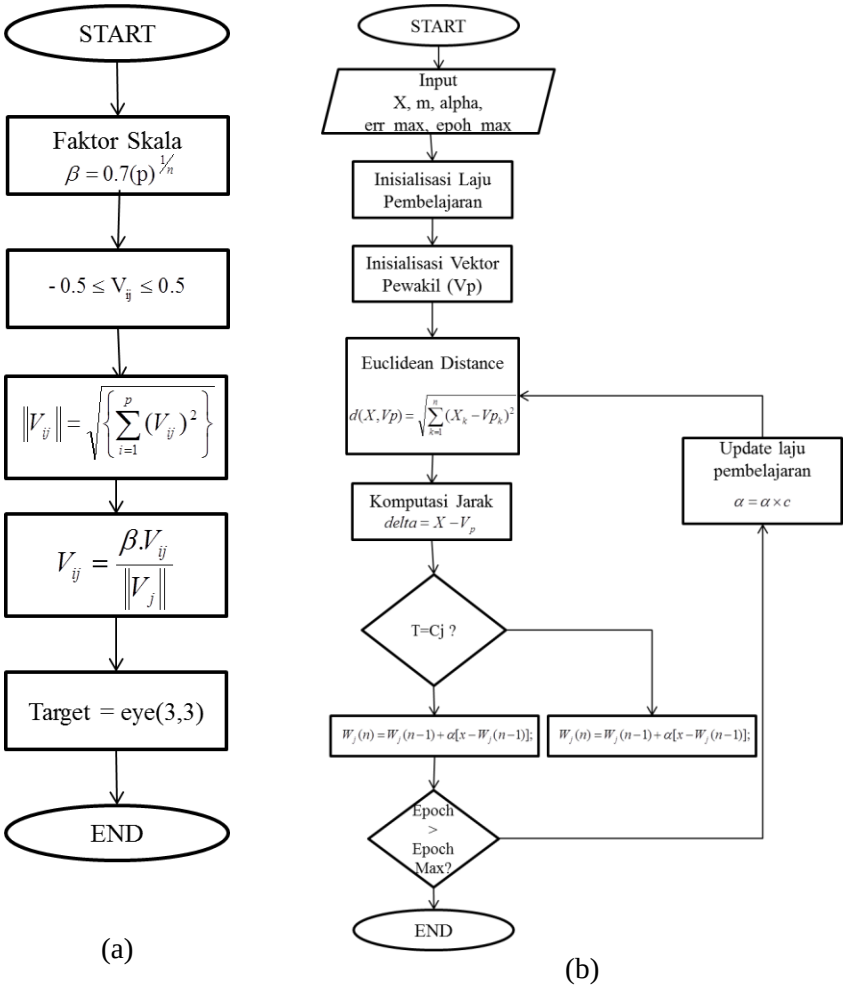
```
%Covariance Matrix
S = cov(Xi_auto)
[P,A] = eig(S)          %P = eigen vector, A = eigen values
A=sum(A,1);             %biar jadi 1 baris, cuma buat plot, ngeliat eigenval yg nilainya gede
figure;
plot(A,'o');
title('eigenvalue');

%Get Scores
A=A(:,n:-1:1);          %dibalik, supaya bisa motong bagian yang penting
P=P(:,n:-1:1);
M = inv(P');
%cek=size(M)
M = M(:,1:L);           %pemotongan eigen vektor
T = (Xi_auto*M);
end
```

Gambar 4. Menghitung *score*

3 PEMBAHASAN

Jaringan syaraf tiruan digunakan untuk memodelkan suatu syaraf biologi pada otak manusia. Penelitian ini menggunakan data *odor* G3-credit.xls. *Learning Vector Quantization* digunakan untuk mengolah data *odor* tersebut sehingga aroma dapat dikenali. Setiap data dari G3-credit.xls memiliki dimensi 15 dengan nilai yang bervariasi. Data yang diperoleh dari G3-credit.xls memiliki nilai yang bervariasi dengan rentang 0.001 hingga 100. Oleh karena itu, LVQ yang diproses akan dilakukan pereduksian dimensi dengan menggunakan metode PCA. Data *training* yang digunakan adalah 70% dari data *odor*. Percobaan ini memiliki *variable* terikat (yang akan diamati) adalah nilai presentase tingkat *Recognition Rate*, nilai *error*, dan *epoch* sedangkan *variable* bebas yang diubah adalah *alpha*, koefisien momentum, dan inisialisasi Vektor pewakil.



Gambar 5. (a) *Flow chart nguyen-widrow*, (b) *Flowchart* ANN *learning vector quantization*

Pada penelitian ini, dilakukan inisialisasi *nguyen-widrow*. Dengan *Flowchart* yang telah diberikan, dibuat program pada *Matlab* dalam bentuk *mfiles*.

4 HASIL PENELITIAN

Untuk percobaan LVQ dilakukan uji pengenalan (*recognition test*) dengan data UCI (G3-Credit). Percobaan dilakukan untuk mengetahui efek perubahan nilai parameter-parameter pelatihan dan penentuan parameter terbaik untuk data tersebut. Pada percobaan ini digunakan data UCI, yaitu G3-credit.xls. Setiap data dari G3-credit.xls memiliki dimensi 15 dengan nilai yang bervariasi. Data yang diperoleh dari G3-credit.xls memiliki nilai yang bervariasi dengan rentang 0,001 hingga 100. Oleh karena itu, akan dilakukan pereduksian dimensi pada LVQ yang diproses dengan menggunakan metode PCA.

Pada percobaan ini *variable* terikat (yang akan diamati) adalah nilai persentase tingkat pengenalan rata-rata (*average Recognition Rate*), nilai *error*, *epoch*, dan waktu pelatihan sedangkan *variable* bebas yang diubah-ubah adalah *alpha*, konstanta, jumlah *epoch*, dan inisialisasi Vektor pewakil. Percobaan dilakukan dengan variasi data *training* 50:50 dan 70:30.

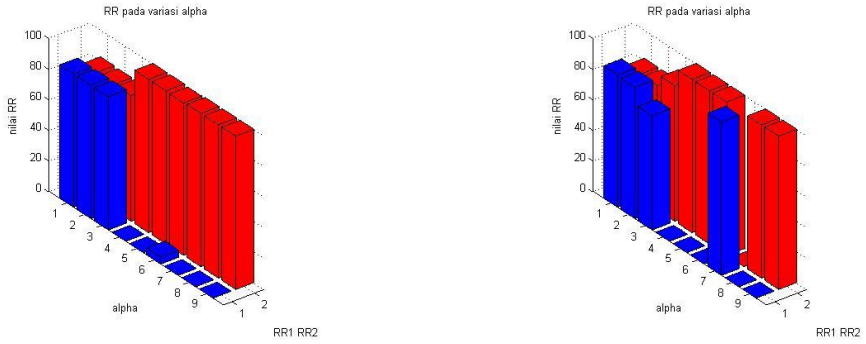
4.1 Pengaruh Perubahan *Alpha* (Laju Pembelajaran)

Pada percobaan ini dilakukan variasi perubahan nilai *alpha* dengan rentang dari 0,1-0,9 dengan interval 0,1. Nilai parameter-parameter yang lain diisi dengan nilai awal.

Percobaan ini dilakukan dengan *ratio* data *training* sebesar 0,5 dan 0,7. Berikut adalah data hasil dari eksperimen.

Tabel 1. Pengaruh perubahan laju pembelajaran

Ratio data *training* = 0.5			*Ratio* data *training* = 0.7		
Alpha	RR1	RR2	*alpha*	RR1	RR2
0,1	86.9707	82.0847	0,1	86.3192	81.1075
0,2	86.9707	81.7590	0,2	85.6678	81.1075
0,3	86.9707	82.0847	0,3	74.5928	88.5993
0,4	0	100.000	0,4	0	100.000
0,5	0	100.000	0,5	0	100.000
0,6	45.603	99.3485	0,6	0	100.000
0,7	0	100.000	0,7	100.000	0
0,8	0	100.000	0,8	0	100.000
0,9	0	100.000	0,9	0	100.000



Ratio data *training* = 0,5 *Ratio* data *training* = 0,7

Gambar 6. Pengaruh perubahan laju pembelajaran

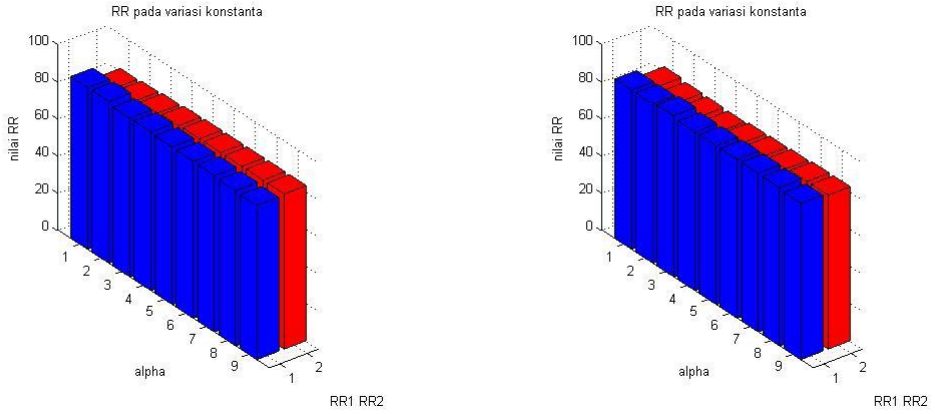
Dari Tabel 1 dan Gambar 6 dapat diamati bahwa nilai laju pembelajaran ini optimal pada nilai 0,2. Pada laju pembelajaran 0,2, nilai *Recognition Rate* cukup stabil pada *ratio* data *training* 0,5 dan 0,7. Terlihat bahwa, semakin tinggi laju pembelajaran, nilai *Recognition Rate* semakin kecil walaupun kecepatan pengolahan data tinggi. Hal ini karena semakin tinggi laju pembelajaran, kecepatan data untuk menuju konvergen lebih cepat, tetapi kurang stabil. Semakin besar *Alpha*, walaupun proses waktu pelatihan semakin cepat, namun *Recognition Rate* akan semakin turun karena *error* dapat masuk ke *local minima.* Karena pada percobaan didapatkan nilai laju pembelajaran optimal adalah 0,2, untuk percobaan selanjutnya akan digunakan nilai *alpha* 0,2.

4.2 Pengaruh Perubahan Konstanta Laju Pembelajaran

Pada percobaan ini dilakukan variasi perubahan nilai konstanta laju pembelajaran dengan rentang dari 0,1-0,9 dengan interval 0,1. Percobaan ini dilakukan dengan *ratio* data *training* sebesar 0,5 dan 0,7. Berikut adalah data hasil dari eksperimen.

Tabel 2. Pengaruh perubahan konstanta laju pembelajaran

Ratio data *training* = 0.5			*Ratio* data *training* = 0.7		
konstanta	RR1	RR2	konstanta	RR1	RR2
0,1	86.9707	81.1075	0,1	85.6678	82.0847
0,2	86.9707	81.7590	0,2	85.6678	81.1075
0,3	85.3420	82.4104	0,3	86.6450	80.7818
0,4	85.3420	83.0619	0,4	85.3420	80.4560
0,5	84.6906	83.3876	0,5	85.3420	81.4332
0,6	84.3648	83.3876	0,6	85.3420	81.7590
0,7	84.0391	83.3876	0,7	85.3420	82.0847
0,8	83.3876	83.3876	0,8	85.0163	82.7362
0,9	82.7362	83.3876	0,9	84.0391	82.7362



Ratio data *training* 0,5 *Ratio* data *training* 0,7

Gambar 7. Pengaruh perubahan konstanta laju pembelajaran

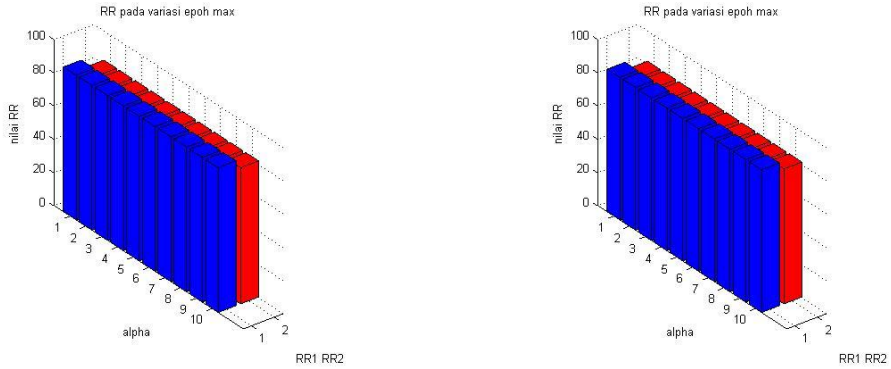
Dari Tabel 2 dan Gambar 7 dapat diamati bahwa nilai konstanta laju pembelajaran ini optimal pada nilai 0,2. Pada konstanta laju pembelajaran 0,2, nilai *Recognition Rate* cukup stabil pada *ratio* data *training* 0,5 dan 0,7. Terlihat bahwa, semakin tinggi konstanta laju pembelajaran, nilai *Recognition Rate* semakin kecil walaupun kecepatan pengolahan data tinggi. Hal ini karena semakin besar konstanta laju pembelajaran, laju pembelajaran akan diperbaharui semakin cepat sehingga kecepatan data untuk menuju konvergen lebih cepat. Semakin besar konstanta, walaupun proses waktu pelatihan semakin cepat, namun *Recognition Rate* akan semakin turun karena *error* dapat masuk ke *local minima.* Karena pada percobaan didapatkan nilai konstanta laju pembelajaran optimal adalah 0,2, untuk percobaan selanjutnya akan digunakan nilai konstanta 0,2.

4.3 Pengaruh Perubahan Jumlah *Epoch*

Pada percobaan ini dilakukan variasi perubahan nilai konstanta laju pembelajaran dengan rentang dari 0,1-0,9 dengan interval 0,1. Percobaan ini dilakukan dengan *ratio* data *training* sebesar 0,5 dan 0,7. Berikut adalah data hasil dari eksperimen.

Tabel 3. Pengaruh perubahan jumlah *epoch*

Ratio data *training* = 0.5			*Ratio* data *training* = 0.7		
Epoch	RR1	RR2	konstanta	RR1	RR2
1.000	86.9707	81.7590	1.000	85.6678	81.1075
2.000	86.9707	81.7590	2.000	85.6678	81.1075
3.000	86.9707	81.7590	3.000	85.6678	81.1075
4.000	86.9707	81.7590	4.000	85.6678	81.1075
5.000	86.9707	81.7590	5.000	85.6678	81.1075
6.000	86.9707	81.7590	6.000	85.6678	81.1075
7.000	86.9707	81.7590	7.000	85.6678	81.1075
8.000	86.9707	81.7590	8.000	85.6678	81.1075
9.000	86.9707	81.7590	9.000	85.6678	81.1075
10.000	86.9707	81.7590	10.000	85.6678	81.1075



Ratio data *training* = 0,5 *Ratio* data *training* = 0,7

Gambar 8. Pengaruh perubahan jumlah *epoch*

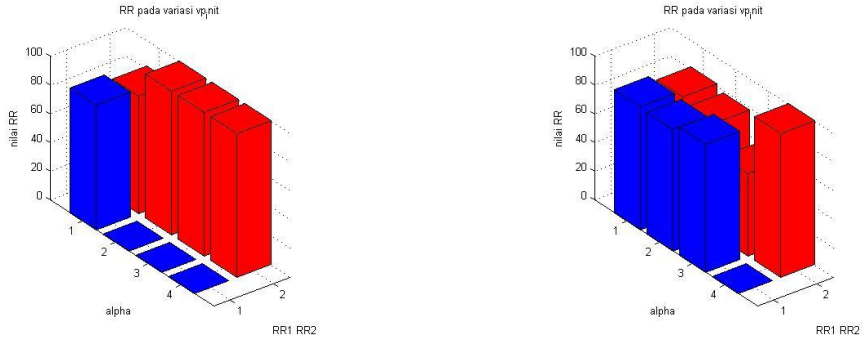
Dari Tabel 3 dan Gambar 8 dapat diamati bahwa jumlah *epoch* maksimum ini optimal pada nilai 1.000. Pada kenyataannya, *epoch* maksimum yang menjadi Standar Internasional bernilai 10.000. Namun, dengan menggunakan parameter-parameter lainnya yang diberikan nilai awal, nilai *error* yang dihasilkan telah mencapai *error* maksimum sehingga pengolahan data terhenti ketika *epoch* kurang dari *epoch* maksimum. Pada percobaan ini, dapat dilihat bahwa *epoch* ketika *error* mencapai nilai *error* maksimum belum mencapai 1.000 sehingga mulai dari *epoch* maksimum 1.000 pengolahan telah terhenti. Dengan demikian, digunakan nilai *epoch* maksimum bernilai 10.000 untuk pengolahan data selanjutnya.

4.4 Pengaruh Perubahan Inisialisasi Vektor Pewakil

Pada percobaan ini dilakukan variasi inisialisasi vektor pewakil. Untuk inisialisasi vektor pewakil dilakukan empat variasi, yaitu inisialisasi vektor pewakil dengan menggunakan salah satu data dari masing-masing kelas, inisialisasi vektor pewakil dengan menggunakan angka *random*, inisialisasi vektor pewakil dengan memilih secara *random* dari keseluruhan data, dan inisialisasi vektor pewakil dengan menggunakan nilai nol. Percobaan ini dilakukan dengan *ratio* data *training* sebesar 0,5 dan 0,7. Berikut adalah data hasil dari eksperimen.

Tabel 4. Pengaruh perubahan inisialisasi vektor pewakil

Ratio data *training* = 0.5			*Ratio* data *training* = 0.7		
Vp_init	RR1	RR2	Vp_init	RR1	RR2
1	86.9707	81.7590	1	85.6678	81.1075
2	0	100.000	2	84.6906	77.8502
3	0	100.000	3	89.2508	57.3290
4	0	100.000	4	0	100.000



Ratio data *training* 0,5 *Ratio* data *training* 0,7

Gambar 9. Pengaruh perubahan inisialisasi vektor pewakil

Dari Tabel 4 dan Gambar 9 dapat diamati bahwa inisialisasi vektor pewakil awal ini optimal pada vp_init 1, yaitu inisialisasi vektor pewakil awal dengan menggunakan salah satu nilai dari data pada masing-masing kelas. Pada inisialisasi vektor pewakil dengan angka *random* dan nilai *random* dari keseluruhan data, terlihat bahwa nilai *Recognition Rate* masih ada pada *ratio* data *training* 0,7. Hal ini karena banyaknya data yang dilatih oleh sistem lebih banyak daripada data pada data *training* 0,5 sehingga data *testing* memiliki nilai *Recognition Rate* yang lebih baik. Pada inisialisasi vektor pewakil dengan nilai nol, nilai *Recognition Rate* pada kelas 1 bernilai nol. Oleh karena itu, digunakan inisialisasi vektor pewakil awal dengan menggunakan data dari masing-masing kelas.

Dari masing-masing percobaan variasi parameter di atas, diperoleh nilai optimal dari masing-masing parameter, yaitu nilai *alpha* (laju pembelajaran) sebesar 0,2, nilai konstanta (konstanta laju pembelajaran) sebesar 0,2, *epoch* maksimum sebesar 1.000, dan inisialisasi vektor pewakil 1, yaitu vektor pewakil dari salah satu data dari setiap masing-masing kelas. Nilai optimal ini digunakan untuk melakukan percobaan selanjutnya, kecuali nilai *epoch* maksimum mengikuti aturan Internasional, yaitu *epoch* maksimum sebesar 10.000.

5 KESIMPULAN

Dari percobaan yang telah dilakukan, diperoleh kesimpulan sebagai berikut:

1. Kenaikan *alpha* berbanding lurus terhadap *training time*.
2. Semakin besar *alpha*, semakin turun nilai *Recognition Rate*.
3. Konstanta laju pembelajaran (α) berbanding lurus dengan *training time*.
4. Konstanta laju pembelajaran yang besar menghasilkan *Recognition Rate* yang rendah.

5. Kenaikan *epoch* berbanding lurus terhadap *training time* dan tidak berpengaruh terhadap *Recognition Rate*.
6. Kenaikan data *training* berbanding lurus terhadap *training time* dan berbanding terbalik dengan *Recognition Rate*.
7. Pengenalan pola dapat dilakukan dengan menggunakan metode ANN *Learning* Vektor *Quantization* (LVQ).

REFERENSI

[1]. Kusumoputro, Benyamin. 2001. Jaringan Neural Buatan, Ed. 1. Jakarta: Universitas Indonesia.
[2]. Marcin Blachnik, Włodzisław Duch. LVQ algorithm with instance weighting for generation of prototype-based rules. Elsevier, 2011: 824-830.
[3]. Mihajlo Grbovic; Slobodan Vucetic. 2009. Learning Vector Quantization with Adaptive Prototype Addition and Removal: IEEE.