Peningkatan Kemampuan Pengenalan Pola dari Jaringan Saraf Tiruan dengan Menggunakan Diskritisasi *Chi2*

Jeffry Hartanto[1], Gregorius Satia Budhi[2]
Program Studi Teknik Informatika
Universitas Kristen Petra
Siwalankerto 121-131
Surabaya
m26409021@john.petra.ac.id[1], greg@petra.ac.id[2]

ABSTRAK

Proses pelatihan *backpropagation* membutuhkan waktu yang cukup lama untuk mencapai tahap konvergen. Salah satu penyebabnya adalah dokumen dalam set data terdiri dari campuran antara bilangan kontinu dan diskrit. dalam hal klasifikasi, sebuah set data akan lebih mudah dibedakan dengan nilai atribut yang berbeda jauh. Metode *Chi2* berhasil untuk menemukan pola dari set data sintetik yang memiliki pola data miring dan pararel. Penggabungan backpropagation dan *Chi2* berhasil mempercepat proses pelatihan dan meningkatkan akurasi klasifikasi. Oleh karena itu, pengujian akan dilanjutkan dengan menggabungkan dua metode tersebut untuk mengklasifikasikan set data dari kasus nyata. Dari hasil pengujian didapatkan kesimpulan bahwa kedua metode tersebut berhasil mempercepat proses pelatihan dan meningkatkan akurasi klasifikasi.

Kata Kunci

Backpropagation, *Chi2*, Diskritisasi dan Jaringan saraf tiruan.

ABSTRACT

Backpropagation training process usually needs quite a long time to reach convergence. One of the reasons is that documents in the data set contains mixed data between continuous and discrete variables. In classification, a data set can be distinguished easier with more varied attribute values. Chi2 method successfully found pattern in an artificial data set which has oblique and parallel data. Moreover, integration between backpropagation and Chi2 can accelerate training process and improve classification accuracy. Therefore, experiments will be done by integrating those processes to classify data set from actual case. Experimental results showed that the two methods could accelerate training process and improve classification accuracy successfully.

Keywords

Backpropagation, Chi2, Discretization and Neural network

1. PENDAHULUAN

Kemampuan komputer terus meningkat seiring dengan berjalannya waktu. Hal ini disebabkan oleh keinginan pembuat aplikasi untuk memaksimalkan kemampuan dari komputer terutama dalam membantu pekerjaan manusia. Terdapat beberapa jenis aplikasi komputer yang membantu pekerjaan manusia, antara lain aplikasi yang berfungsi untuk mengenali suatu objek, suara ataupun tulisan dengan klasifikasi, aplikasi yang menghasilkan suatu prediksi terhadap sebuah permasalahan dan aplikasi untuk menyelesaikan permasalahan yang bersifat *non-linear*. Salah satu ilmu komputer yang digunakan untuk mengatasi masalah diatas adalah jaringan saraf tiruan yang berdasar pada jaringan saraf manusia.

Jaringan saraf manusia terdiri atas sel-sel saraf (*neuron*) yang saling berhubungan. *Input* dari jaringan saraf itu sendiri adalah 5 indera manusia yaitu pendengaran, penciuman, penglihatan, perasa dan peraba. Saat salah satu dari kelima indera diberikan rangsangan maka, bagian sel yang disebut dendrit akan menerima rangsangan tersebut dan tiap *neuron* akan menyalurkan rangsangan tersebut hingga ke otak. Kemudian, otak akan mengeluarkan *output* berupa tindakan atau respon atas rangsangan yang diterima. Berdasarkan sistem kerja dari jaringan saraf manusia, muncullah ilmu komputer yang membuat tiruan dari jaringan saraf manusia.

Sama halnya dengan jaringan saraf manusia, jaringan saraf tiruan terdiri dari beberapa *input neuron*, *hidden neuron* dan *output neuron*. Tiap *neuron* pada jaringan saraf tiruan dihubungkan dengan beberapa *weight* dengan *neuron* lainnya. Akan tetapi, struktur dari jaringan saraf tiruan tidak sekompleks jaringan saraf manusia. Terdapat berbagai metode dalam jaringan saraf tiruan seperti *backpropagation*, *self-organized mapping*, *hopfield*, *mexican hat* dan masih banyak lagi. *Backpropagation* merupakan metode *supervised training* yang cukup populer dibandingkan dengan algoritma jaringan saraf tiruan lainnya. Namun, salah satu kelemahan dari *backpropagation* adalah waktu *training* yang dibutuhkan cukup lama untuk dapat mengklasifikasikan suatu masalah yang kompleks.

Oleh karena itu, terdapat penelitian [7] yang berfokus pada peningkatkan kecepatan proses *training* dari sebuah algoritma klasifikasi. Ide yang mendasari penelitian tersebut adalah menambahkan suatu nilai yang memperjelas perbedaan data sampel satu dengan yang lainnya atau biasa disebut *cut-off*. Dengan memperjelas perbedaan antar data sampel, maka proses klasifikasi pun akan menjadi lebih cepat. Akan tetapi, pada penelitian tersebut penentuan *cut-off* dilakukan secara manual. Selain itu, *cut-off* yang dihasilkan memiliki interval yang sama. Untuk mengatasi keterbatasan tersebut, dalam skripsi ini akan digunakan algoritma *Chi2* untuk menemukan *cut-off* secara otomatis dan dengan interval yang sesuai dengan pola pada data sampel.

2. TINJAUAN PUSTAKA

2.1 Data Sintetik

Karena salah satu tujuan dari skripsi ini adalah menemukan perpotongan pada data, maka sudah disediakan beberapa pola data yang dapat digunakan. Terdapat 2 macam perpotongan/pola data yaitu, perpotongan data secara axis pararel dan miring. Semua data sintetis tersebut digunakan dengan tujuan untuk melihat kekuatan dari arsitektur yang dihasilkan. Dalam skripsi ini terdapat 5 jenis data set, yaitu *chess data set, cross data set, cube data set, x data set* dan *two-triangle data set* [7]**.** Setiap *data set* diklasifikasikan menjadi 2 kelas, dimana lingkaran mewakili data yang berada di kelas pertama, sedangkan wajik mewakili data yang berada di kelas kedua. Untuk lebih jelasnya, Gambar 1 merupakan contoh distribusi data dari kelima *data set*.

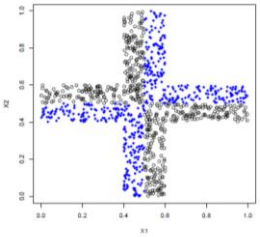


Fig. 2. Cross data set

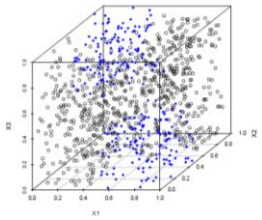


Fig. 4. Cube data set

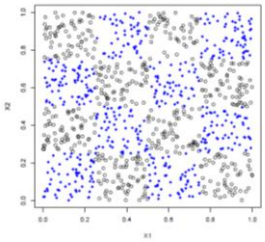


Fig. 3. Chess data set

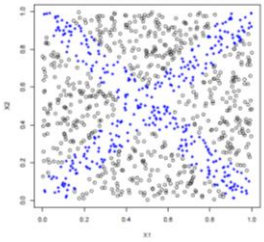


Fig. 5. X data set

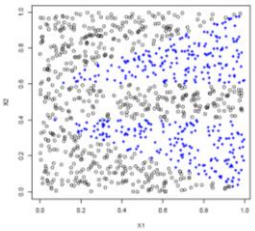


Fig. 6. Two-triangle data set

Gambar 1. Kelima data sintetik yang digunakan

2.2 *Chi2*

Algoritma *Chi2* [3] adalah sebuah algoritma yang menggunakan $\chi 2$ statistik untuk mendiskritkan atribut yang bernilai numerik. Maka dari itu, algoritma ini cukup efektif jika digunakan dalam penyeleksian fitur-fitur penting dari sekelompok atribut numerik. Dengan menggunakan fitur-fitur yang relevan, maka algoritma ini dapat mempercepat proses pelatihan dan meningkatkan akurasi prediksi dari algoritma klasifikasi pada umumnya. Ditambah lagi, banyak algoritma klasifikasi yang membutuhkan dan bekerja lebih baik pada data training yang bersifat diskrit.

Dalam penggunaannya, algoritma *Chi2* dibagi menjadi 2 fase. Fase pertama dimulai dengan nilai signifikansi yang cukup tinggi, misal 0.5, pada semua atribut untuk di diskritisasi. Proses penggabungan data akan terus dilakukan selama nilai $\chi 2$ tidak melebihi nilai yang ditentukan dari signifikansi (0.5, menghasilkan nilai 0.455 dengan *degree of freedom* bernilai 1). Fase ini akan terus diulang dengan mengurangi nilai signifikansi, hingga angka *inconsistency* dari data yang di diskritisasi melebihi batasan yang ditentukan. Formula untuk menghitung nilai $\chi 2$ dapat dilihat pada persamaan 1.

$$\chi^2 = \sum_{i=1}^{2}\sum_{j=1}^{k}\frac{(A_{ij} - E_{ij})^2}{E_{ij}} \quad (1)$$

Dimana,

k = jumlah dari klasifikasi,

A_{ij} = jumlah dari pola pada interval ke-i, klasifikasi ke-j,

R_i = jumlah dari pola pada interval ke-i = $\sum_{j=1}^{k} A_{ij}$,

C_j = jumlah dari pola pada klasifikasi ke-j = $\sum_{i=1}^{2} A_{ij}$,

N = jumlah total dari seluruh pola = $\sum_{i=1}^{2} R_i$,

E_{ij} = jumlah pola yang diharapkan dari A_{ij} = $R_i * C_j/N$, jika R_i atau C_j bernilai 0, maka ubah nilai E_{ij} menjadi 0.1.

Fase kedua adalah tahap optimasi dari fase pertama. Perbedaan yang paling terlihat adalah pada perhintungan *inconsistency*, dimana pada fase 2 perhitungan dilakukan setelah semua atribut melalui proses penggabungan. Sedangkan pada fase pertama, nilai *inconsistency* dihitung pada akhir proses diskritisasi tiap atribut. Fase kedua akan terus diulang, hingga tidak ada nilai dari atribut yang dapat di diskritisasi atau digabungkan. *Inconsistency* terjadi saat terdapat dua atau lebih sampel yang seluruh atributnya memiliki nilai yang sama namun, terdapat pada kelompok yang berbeda.

2.3 *Error Back-propagation Training*

Backpropagation, singkatan dari "*backward propagation of errors*", adalah sebuah metode pelatihan yang umum digunakan dalam jaringan saraf tiruan. Metode ini cukup umum digunakan karena kemampuannya dalam menyelesaikan masalah-masalah yang bersifat *non-linear*. *Backpropagation* dilatih dengan metode *supervised learning*, yang berarti membutuhkan sebuah target yang ingin dihasilkan untuk tiap sampelnya. Berikut ini merupakan langkah-langkah yang harus dilakukan untuk dapat menerapkan algoritma *backpropagation* [4]:

1. *Initialization*

Tentukan nilai dari semua *weight* dan bias yang berada pada suatu *range* tertentu secara *random*:

$$(-1, +1) \quad (2)$$

2. *Activation*

Hitung nilai *output* dari semua *neuron* yang terdapat pada *hidden layer* dengan menggunakan Persamaan 3:

$$y_j(p) = sigmoid\left[\sum_{i=1}^{n} x_i(p) \times w_{ij}(p) - \theta_j\right] \quad (3)$$

dimana *n* adalah jumlah *input neuron* dari tiap *neuron j* pada *hidden layer*.

Hitung nilai *output* dari semua *neuron* yang terdapat pada *output layer* dengan menggunakan Persamaan 4:

$$y_k(p) = sigmoid\left[\sum_{j=1}^{m} x_{jk}(p) \times w_{jk}(p) - \theta_k\right] \quad (4)$$

dimana *m* adalah jumlah *input neuron* dari *neuron k* pada *output layer*. Kemudian untuk *sigmoid activation function* adalah sebagai berikut:

$$Y^{sigmoid} = \frac{1}{1 + e^{-S}} \quad (5)$$

dimana *S* adalah nilai dari persamaan 3 atau 4, *p* menandakan jumlah iterasi dan θ adalah nilai bias yang terdapat pada tiap *neuron*.

3. *Weight training*

Perbarui nilai *weight* dari *output layer* hingga *input layer* sesuai dengan *error* yang dihasilkan dari selisih antara nilai target dengan nilai *neuron* pada *output layer*. Hitung *error gradient* dari semua *neuron* pada *output layer* dengan menggunakan Persamaan 6 dan 7:

$$\delta_k(p) = y_k(p) \times [1 - y_k(p)] \times e_k(p) \quad (6)$$

$$e_k(p) = \left[y_{d,k}(p) - y_k(p)\right] \quad (7)$$

dimana $y_{d,k}(p)$ adalah nilai yang menjadi target untuk *output neuron* $y_k(p)$. Hitung *weight corrections* dengan menggunakan Persamaan 8:

$$\Delta w_{jk}(p) = \alpha \times y_j(p) \times \delta_k(p) \quad (8)$$

Perbarui semua nilai *weight* yang terhubung dengan *output neuron* dengan menggunakan Persamaan 9:

$$w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p) \quad (9)$$

Hitung *error gradient* dari semua *neuron* yang terdapat pada *hidden layer* dengan menggunakan Persamaan 10:

$$\delta_j(p) = y_j(p) \times [1 - y_j(p)] \times \sum_{k=1}^{l} \delta_k(p) \times w_{jk}(p) \quad (10)$$

Hitung *weight corrections* dengan menggunakan persamaan berikut:

$$\Delta w_{ij}(p) = \alpha \times x_i(p) \times \delta_j(p) \quad (11)$$

Perbarui semua nilai *weight* yang terhubung dengan *hidden neuron* dengan menggunakan persamaan berikut:

$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p) \quad (12)$$

4. *Iteration*

Tambahkan nilai *p* sebanyak 1, kemudian ulangi dari langkah ke-2. Proses ini akan terus berjalan hingga mencapai nilai *sum of squared error* yang sudah ditentukan. Untuk menghitung nilai SSE dapat menggunakan persamaan berikut [6]**:**

$$E(w,v) = \frac{1}{2} \times \sum_{i=1}^{k} \sum_{p=1}^{C} (S_{pi} - t_{pi})^2$$

dimana, k adalah jumlah sampel data, C adalah jumlah *output neuron*, t_{pi} adalah target untuk sampel data ke-*i* pada *output neuron* ke p, dimana p = 1, 2, 3, …, C. S_{pi} adalah *output* yang dihasilkan dari sebuah jaringan pada *output neuron* ke p. S_{pi} dapat dihasilkan dengan menggunakan Persamaan 3 kemudian dilanjutkan menggunakan Persamaan 4. Selama proses pelatihan masih berjalan, nilai dari *weight* akan terus berubah-ubah seiring dengan bertambahnya jumlah iterasi. Perubahan tersebut diharapkan menghasilkan nilai *sum of squared error* dibawah nilai yang sudah ditentukan atau *minimum error*.

2.4 K-*fold-cross-validation*

Algoritma ini biasanya digunakan untuk mengukur seberapa besar kemampuan generalisasi dari jaringan saraf tiruan terhadap data yang bersifat independen [5]. Jika kemampuan generalisasi tidak diperhitungkan, maka besar kemungkinan struktur jaringan saraf tiruan yang dihasilkan akan terlalu fokus pada data *training* atau *over-fitting*. Yang kemudian akan menyebabkan kemampuan prediksi dari jaringan saraf tiruan menurun. Algoritma ini bekerja dengan membagi data sampel menjadi k *subsets* atau biasa disebut k *validation sets*. Pada umumnya k bernilai 10, berikut adalah *pseudo code* dari algoritma ini:

1. Bagi data *training* menjadi Lk *subsets*, yang tiap *subset*-nya memiliki jumlah data yang hampir sama.
2. Untuk tiap iterasi k=1, 2,…, k. Aturlah sehingga, Vk = Lk dan Tk = D - Lk, dimana Vk adalah *validation set*, Lk adalah *subset* ke-k, D adalah data *training* dan Tk adalah data yang digunakan untuk *training*.
3. Setelah menggunakan Tk untuk proses *training*, gunakan Vk untuk proses *testing*. Kemudian bandingkan nilai SSE dari keduanya, jika SSE dari Vk melebihi Tk maka hentikan pelatihan.

2.5 *Pruning*

Pruning adalah proses penghapusan komponen dari sebuah jaringan, yang nantinya menghasilkan arsitektur jaringan yang seminimal mungkin untuk menyelesaikan pola/klasifikasi dari suatu masalah [6]. Proses ini menjadi sangat penting, karena salah satu kesulitan dalam menggunakan struktur jaringan *feedforward* adalah menentukan jumlah *hidden unit* yang optimal sebelum proses *training* dimulai. Masalah diatas timbul dikarenakan, saat sebuah jaringan memiliki terlalu banyak *hidden unit*, maka jaringan tersebut akan terlalu fokus pada data *training* sehingga memiliki generalisasi yang lemah terhadap data *testing*. Demikian pula sebaliknya, jika sebuah jaringan memiliki terlalu sedikit *hidden unit*, maka jaringan tersebut tidak dapat mengklasifikasi data dengan baik.

Terdapat 2 pendekatan yang diusulkan untuk menemukan jumlah *hidden unit* yang optimal. Pertama, dengan memulai dari jumlah *hidden unit* yang banyak kemudian menghilangkan *hidden unit* yang tidak diperlukan [1]. Kedua, dengan memulai dari jumlah *hidden unit* yang sedikit kemudian menambahkan *hidden unit* baru untuk meningkatkan akurasi dari jaringan [2]. Dalam skripsi ini proses *pruning* melalui 3 tahapan, yaitu *pruning input neurons*, *pruning hidden neurons* dan *pruning weights*.

2.5.1 *Pruning Input/Hidden Neurons*

Proses *pruning input/hidden neurons* bertujuan untuk menghilangkan *input/hidden neuron* yang tidak berguna untuk proses klasifikasi data. Metode *trial and error* [8] digunakan untuk mencari *input/hidden neuron* mana saja yang bisa dihilangkan dari jaringan. Pemilihan urutan antara *pruning hidden* atau *input* tidak terlalu mempengaruhi hasil akhir dari proses keseluruhan *pruning*. Langkah-langkah yang dilakukan dalam proses *pruning input unit* adalah sebagai berikut [7]**:**

1. Kelompokan IH kedalam 2 himpunan: P dan Q, dimana IH sama dengan *input neuron* atau *hidden neuron*, P adalah IH yang masih ada di dalam jaringan dan Q adalah IH yang dikeluarkan dari jaringan. Pertama-tama, himpunan P berisi semua IH yang ada di jaringan dan Q adalah himpunan kosong.
2. Simpan replika dari struktur jaringan saat ini.
3. Cari IH yang terdapat pada himpunan P, sehingga saat semua koneksi/*weights* yang berhubungan dengan IH tersebut diberi nilai 0, perubahan tersebut memberikan efek yang paling minim terhadap akurasi dari jaringan tersebut.
4. Ubahlah semua koneksi yang terhubung dengan IH yang dipilih pada langkah ketiga, kemudian lakukan *training* kepada jaringan yang sudah di *pruning*.

Jika akurasi yang didapat dari langkah keempat masih memuaskan, maka pindahkan IH yang dipilih pada langkah ketiga kedalam himpunan Q. Setelah itu, ulangi lagi proses diatas mulai dari langkah kedua. Jika tidak, maka kembalikan struktur jaringan ke struktur yang sudah disimpan pada langkah kedua. Selama himpunan P ≠ himpunan Q maka kembali ke langkah kedua, selain itu berhenti.

2.5.2 *Pruning Weights*

Proses *pruning* dilanjutkan dengan menghilangkan koneksi/*weight* yang tidak membantu struktur jaringan dalam proses klasifikasi. Koneksi yang akan di *pruning* dipilih dengan mencari koneksi yang nilainya paling mendekati 0. Syarat tersebut menjadi acuan, dikarenakan setiap koneksi yang memiliki nilai mendekati 0 merupakan koneksi yang kurang berguna bagi jaringan. Untuk optimasi waktu, proses ini dilakukan setelah *pruning input* dan *hidden neurons*. Berikut merupakan langkah-langkah yang dilakukan dalam proses *pruning* koneksi/*weight*:

1. Kelompokan koneksi kedalam 2 himpunan: P dan Q, dimana P adalah koneksi yang masih ada di dalam jaringan dan Q adalah koneksi yang dikeluarkan dari jaringan. Pertama-tama, himpunan P berisi semua koneksi yang ada di jaringan dan Q adalah himpunan kosong.
2. Simpan replika dari struktur jaringan saat ini.
3. Carilah koneksi yang memiliki nilai terendah dengan persamaan 14.

$$\min |weight_{wv}| \, , weight_{wv} \in (w \cup v) \quad (14)$$

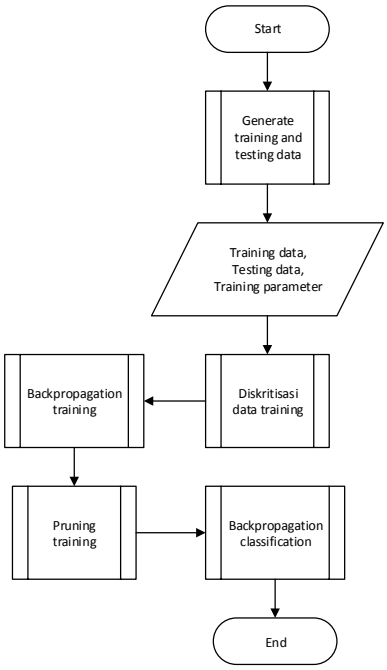
dimana, w adalah semua koneksi yang menghubungkan *input neuron* dengan *hidden neuron*, sedangkan v adalah semua koneksi yang menghubungkan *hidden neuron* dengan *output neuron*.

4. Ubahlah nilai dari koneksi yang dipilih pada langkah ketiga menjadi 0, kemudian lakukan *training* kepada jaringan yang sudah di *pruning*.
5. Jika akurasi yang didapat dari langkah keempat masih memuaskan, maka pindahkan koneksi yang dipilih pada langkah ketiga kedalam himpunan Q. Setelah itu, ulangi lagi proses diatas mulai dari langkah kedua. Jika tidak, maka kembalikan struktur jaringan ke struktur yang sudah disimpan pada langkah kedua. Selama himpunan P ≠ himpunan Q maka kembali ke langkah kedua, selain itu berhenti.

3. Desain Sistem

Untuk dapat mengoperasikan perangkat lunak ini, ada langkah-langkah yang harus dilakukan. Pertama, jika belum memiliki data maka pengguna dapat memilih modul *generate input file*, dimana pada modul tersebut terdapat beberapa pilihan pola data. Kemudian, dilanjutkan dengan membagi data yang sudah disimpan pada *file* menjadi 2 bagian, yaitu data *training* dan data *testing*. Setelah itu, pengguna dapat mengatur beberapa parameter yang berhubungan dengan proses *training* (jumlah *hidden neuron*, jumlah iterasi, *learning rate* dan nilai *threshold*).

Untuk proses *training*, pengguna dapat memilih untuk menambahkan variabel diskrit secara manual atau otomatis. Setelah proses *training* berakhir, pengguna dapat lanjut ke proses *pruning*, dimana arsitektur yang dihasilkan dari proses *training* akan dibuat menjadi lebih sederhana. *Output* dari program ini adalah arsitektur jaringan saraf tiruan, yang nantinya pengguna dapat menyimpan arsitektur tersebut ke dalam *file*. Untuk lebih jelasnya dapat dilihat pada Gambar 2.



Gambar 2. Diagram alir garis besar sistem kerja perangkat lunak

4. Hasil Eksperimen

Pengujian sistem secara garis besar dapat dibagi menjadi 3 bagian besar, yaitu pengujian proses diskritisasi, training dan pruning. Proses diskritisasi akan diuji dengan mengganti nilai

inconsistency untuk membandingkan *output* yang dihasilkan. Untuk pengujian proses *training* akan dilihat efek yang diberikan dengan menggunakan atau tidak menggunakan tambahan variabel diskrit. Sedangkan untuk proses *pruning* akan diuji efek yang diberikan dengan menggunakan atau tidak menggunakan tambahan varibel diskrit.

Dataset yang akan digunakan dalam pengujian merupakan 5 *dataset* sintetik yang sudah terdapat pada aplikasi, yaitu *chess dataset, cross dataset, cube dataset, two-triangle dataset* dan *x dataset*. Selain itu, akan dilakukan pengujian terhadap 3 macam *dataset* yang diunduh dari *UCI Machine Learning Repository*, yaitu *wdbc dataset, wpbc dataset* dan *breast-cancer dataset*. Dimana setiap *dataset* memiliki perpaduan tipe atribut antara lain atribut bersifat diskrit, atribut bersifat kontinu dan atribut bersifat campuran antara diskrit dan kontinu.

4.1 *Chi2* dengan *Inconsistency* sebesar 5%

Proses pengujian yang akan dilakukan adalah pengamatan terhadap *output* yang dihasilkan dengan nilai *inconsistency* sebesar 5% dari algoritma *Chi2*. *Inconsistency* terjadi saat paling sedikit terdapat 2 data sampel yang semua atribut memiliki nilai yang sama, namun berada pada 2 kelompok yang berbeda. Dari tiap pengujian akan ditampilkan *output* atau interval yang dihasilkan oleh algoritma *Chi2* untuk kelima data sintetik. Dimana, X_1, X_2,…, X_n merupakan atribut dari suatu sampel.

Tabel 1. *Cut-offs chess*, *cross* dan *cube dataset* dengan 5% *inconsistency*

***Dataset* (*Inconsistency* = 5%)**						
Chess		***Cross***		***Cube***		
X_1	X_2	X_1	X_2	X_1	X_2	X_3
0.0006	0.0001	0.4004	0.4002	0.0012	0.0005	7.58-06
0.2545	0.2561	0.4993	0.4990	0.4797	0.5138	0.5001
0.5116	0.5019					
0.7552	0.7467					

Tabel 2. *Cut-offs two-triangle* dan *X dataset* dengan 5% *inconsistency*

***Dataset* (*Inconsistency* = 5%)**			
Two-triangle		***X***	
X_1	X_2	X_1	X_2
0.0005	0.0013	0.0005	0.0004
0.1938	0.0836	0.0797	0.0798
0.3028	0.2644	0.1682	0.1888
0.3740	0.4002	0.2456	0.2486
0.3804	0.6528	0.3088	0.3197
0.5316	0.7777	0.3717	0.3995
0.7282		0.6062	0.4826
0.8032		0.6628	0.5628
0.8093		0.7386	0.6495
0.9453		0.7982	0.7586
		0.8756	0.8343
			0.8864

Dengan nilai *inconsistency* sebesar 5%, algoritma *Chi2* berhasil menemukan nilai *cut-off* yang mewakili perpotongan data terutama untuk *data set* yang bersifat axis pararel, seperti *chess data set, cross data set* dan *cube data set*. Sedangkan untuk *data set* yang bersifat campuran terlihat bahwa *cut-off* dari perpotongan axis-pararel masih bisa didapatkan. Untuk *data set* dengan perpotongan data bersifat *oblique* terlihat masih banyak interval yang dihasilkan, seperti yang sudah diperkirakan sebelumnya.

4.2 Akurasi yang Dihasilkan Tiap *Data Set* dengan Tanpa Diskritisasi dan Diskritisasi *Chi2*

Pengujian yang akan dilakukan adalah membandingkan akurasi yang didapat dengan berbagai metode yang digunakan. Sedangkan untuk algoritma *Chi2* akan digunakan *inconsistency* sebesar 0% dan 5%. Percobaan akan dilakukan dengan 1 *hidden layer*, 8 *hidden unit*, *learning rate* sebesar 0.1 selama 50 iterasi.

Tabel 3. Perbandingan akurasi antara tanpa diskritisasi dengan *inconsistency* algoritma *Chi2* 0% dan 5%

Dataset	**Tipe**	**Akurasi (dalam %)**		
		Tanpa diskritisasi	***Chi2* 0%**	***Chi2* 5%**
Chess	***Training***	52.62	98.92	98.15
	Testing	50.90	95.78	96.38
Cross	***Training***	52.30	99.97	99.87
	Testing	51.38	98.64	100.0
Cube	***Training***	86.12	99.55	99.35
	Testing	83.90	97.52	99.16
Two-triangle	***Training***	90.17	99.47	98.75
	Testing	91.12	96.14	95.48
X	***Training***	55.22	97.87	95.73
	Testing	52.64	92.74	89.34
Breast cancer	***Training***	97.22	97.15	95.62
	Testing	98.06	97.53	98.59
WDBC	***Training***	89.09	98.83	94.83
	Testing	89.11	92.56	94.77
WPBC	***Training***	76.04	88.75	83.95
	Testing	76.92	70.51	75.12

Jika diperhatikan dari hasil pengujian pada Tabel 5, Jika diperhatikan dari hasil pengujian pada Tabel 5.13, dapat dilihat bahwa dengan *inconsistency* 0% akurasi untuk proses *training*

selalu lebih baik dibandingkan dengan *inconsistency* 5%. Jika dilihat dari akurasi proses *testing* untuk *dataset* asli dan *dataset* dengan perpotongan *axis-parallel*, didapatkan hasil yang sebaliknya dimana akurasi *inconsistency* 5% lebih baik dari pada *inconsistency* 0%. Sedangkan untuk *dataset* dengan perpotongan *oblique* akurasi yang paling optimal didapatkan dengan menggunakan inconsistency sebesar 0%. Oleh karena itu, dapat disimpulkan bahwa setiap *dataset* memiliki nilai *inconsistency* yang berbeda-beda untuk dapat mencapai akurasi yang optimal.

4.3 Pengamatan Informasi yang Dihasilkan Tiap *Dataset*, dengan Algoritma *Chi2* dalam Proses *Pruning*

Pengujian ini bertujuan untuk mengetahui apakah proses *pruning* berhasil mempertahankan *input neuron* yang mewakili *cut-off dataset*, yang didapat dari diskritisasi algoritma *Chi2*. Nilai *inconsistency* yang akan digunakan dalam pengujian ini sebesar 5%. Proses *pruning* akan dilakukan sebanyak 5 kali, yang kemudian informasi yang akan ditampilkan merupakan rata-rata yang dihasilkan dari 5 kali pengujian tersebut. Pengujian proses *pruning* dilakukan kepada kelima *dataset* sintetik dengan parameter sebagai berikut, 10 *epochs* untuk *retraining*, *learning rate* sebesar 0.1 dan batas penurunan akurasi sebesar 5%. Tabel 5.17 dan 5.18 merupakan tabel informasi dari proses *pruning* dengan diskritisasi manual. Dimana, H adalah jumlah *hidden neuron*, I adalah jumlah *input neuron* dan C adalah jumlah koneksi yang masih aktif pada jaringan.

Tabel 4. Hasil *pruning* tiap *dataset* sintetik dengan 5% *inconsistency*

Dataset	**Sebelum *Pruning***			**Sesudah *Pruning***			**Akurasi**	
	H	**I**	**C**	**H**	**I**	**C**	**Train**	**Test**
Chess	8	8	48	2	6	16	97.75	96.20
Cross	8	4	80	2	2	7.8	99.37	99.09
Cube	8	6	64	2	3	10.2	97.57	97.55
Two-triangle	8	16	144	2.4	3	10.6	96.87	95.97
X	8	25	216	3	15.8	45.4	91.01	86.00

Hasil algoritma *Chi2* yang di *pruning* pun berhasil menyisakan *cut-off* dari *dataset*. Seperti yang dapat dilihat dari jumlah *neuron* yang dihasilkan di tiap *dataset* dengan 5% *inconsistency*. *Chess dataset* memiliki 6 *input neuron* yaitu 0.25, 0.5 dan 0.75 untuk tiap atribut. *Cross* dan *cube dataset* memiliki 2 dan 3 *input neuron* secara berurutan, dimana tiap atribut merupakan hasil perpotongan data pada interval 0.5. Untuk *two-triangle dataset*, *cut-off* yang ditemukan adalah 0.4 dan *input* asli dari *dataset* karena *dataset* ini merupakan campuran dari *oblique* dan pararel. Sedangkan untuk *X dataset* masih tersisa banyak *input neuron* karena bentuknya yang *oblique*.

5. Kesimpulan

Dari hasil pengujian dapat disimpulkan beberapa hal, yaitu algoritma *Chi2* berhasil menemukan *cut-off* yang bersifat axis pararel dari *data set*. Sedangkan untuk *data set* yang memiliki *cut-off* bersifat *oblique*, maka interval yang dihasilkan masih cukup banyak. Algoritma *Chi2* dapat menggabungkan interval yang tidak berperan penting dari tiap atribut untuk proses klasifikasi dari suatu data sampel. Dengan demikian dapat disimpulkan pula bahwa dengan menambahkan variabel diskrit pada *input layer*, penurunan nilai SSE dapat dipercepat. Hasil *pruning* menunjukkan bahwa variabel diskrit yang mewakili *cut-off* dari *data set* yang memiliki *cut-off* bersifat *axis-parallel* dapat menggantikan peranan dari atribut asli. Sedangkan untuk *data set* yang bersifat *oblique* atau campuran proses *pruning* tidak menghilangkan atribut asli dari *data set* tersebut dan meninggalkan beberapa variabel diskrit yang berguna.

Beberapa percobaan yang dapat dilakukan kedepannya adalah melakukan eksperimen dalam menggabungkan diskritisasi *input* dengan jaringan saraf tiruan untuk klasifikasi. Melakukan percobaan pada metode yang sudah ada ini ke lebih banyak data asli atau kasus nyata.

6. REFERENCES

[1] Chung, F. L., & Lee, T. (1992). *A node pruning algorithm for backpropagation networks*. *Int. J. Neural System* 3(3), 301-314.

[2] Frean, M. (1990). *The upstart algorithm: A method for constructing and training feedforward neural networks*. *Neural Computation* 2(2), 198-209.

[3] Liu, H. & Setiono, R. (1995, November). *Chi2: Feature selection and discretization of numeric attributes*. *In Proc. of the 7th International Conference on Tools with Artificial Intelligence*, Washington D.C., pp. 388-391.

[4] Negnevitsky, M. (2005). *Artificial intelligence: A guide to intelligence systems* (2nd Ed.). New York: Addison Wesley.

[5] Refaeilzadeh, P., Tang, L., Liu, H. (2009). *Cross Validation*. (Liu, L. & Tamer Ozsu M., Eds.) *Encyclopedia of Database Systems*. Springer, pp. 532-538.

[6] Setiono, R. (1997). *A penalty-function approach for pruning feedforward neural networks*, *Neural Computation*, vol. 9, no. 1, pp. 185-204, 1997.

[7] Setiono, R. & Seret, A. (2012, November). *Discrete variable generation for improved neural network classification*. *In Proc. of the 43rd annual meeting of the Decision Sciences Institute*. San Fransisco, USA.

[8] Young, H., P. (2009, March). *Learning by trial and error*. *Games and Economic Behavior*, vol.65, pp. 626-643.