KLASIFIKASI NON LINIER

Anggun Fitrian Isnawati, 06244-S2 TE
Sulistyaningsih, 05912-S2 TE
Rintania Ellyati Nuryaningsih, 06245-S2 TE
Jurusan Teknik Elektro dan Teknologi Informasi FT UGM,
Yogyakarta

4.1 PENDAHULUAN

Pada bab sebelumnya kita membahas mengenai klasifikasi linier yang digambarkan oleh fungsi diskriminan linier (hyperplanes) g (x). Dalam kasus dua kelas sederhana, dapat dilihat bahwa algoritma perseptron menghitung bobot linier dari fungsi g (x), asalkan kelas linier dipisahkan. Untuk klasifikasi linier kelas-kelas yang dapat dipisahkan secara non-linier dirancang secara optimal, misalnya, dengan meminimumkan kuadrat kesalahan. Dalam bab ini kita akan menghadapi masalah yang tidak dapat dipisahkan secara linier dan bahkan secara optimal, tidak dapat mengakibatkan kinerja yang memuaskan. Desain penggolong nonlinier muncul sebagai kebutuhan yang tidak dapat terelakkan.

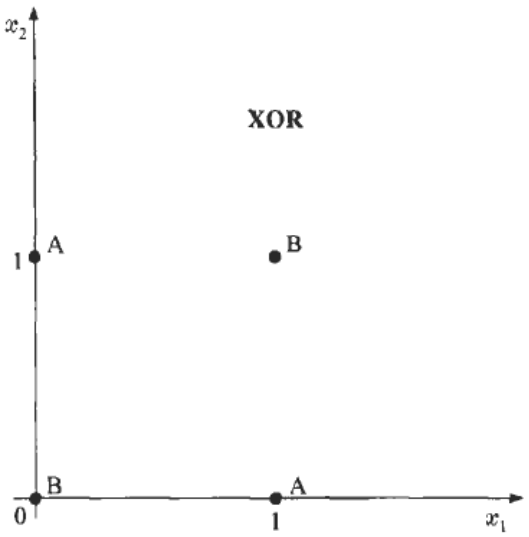
4.2 PERMASALAHAN XOR

Untuk mencari masalah yang dapat dipisahkan secara non linier tidak perlu repot lagi. Fungsi Boolean Eksklusif OR (XOR) yang sudah terkenal merupakan contoh khusus dari permasalahan tersebut. Fungsi Boolean dapat diartikan sebagai klasifikasi tugas. Tergantung dari nilai input data biner $x=[x1,x2,x3,,,xn]^T$ maka output akan bernilai 0 atau 1, dan x diklasifikasikan ke dalam salah satu dari dua kelas A(1) atau B(0). Tabel kebenaran yang sesuai untuk operasi XOR adalah ditunjukkan pada Tabel 4.1.

Tabel 4.1 Tabel Kebenaran untuk Permasalahan XOR

x_1	x_2	XOR	Class
0	0	0	B
0	1	1	A
1	0	1	A
1	1	0	B

Gambar 4.1 menunjukkan posisi dalam ruang kelas. Hal ini jelas terlihat dari gambar tersebut tidak ada satu garis lurus yang memisahkan dua kelas. Sebaliknya, dua fungsi Boolean yang lainnya yaitu AND dan OR, dapat dipisahkan secara linear. Tabel kebenaran yang sesuai untuk operasi OR dan AND ditunjukkan dalam Tabel 4.2 dan posisi kelas masing-masing dalam ruang dua dimensi ditunjukkan pada Gambar 4.2a dan 4.2b. Gambar 4.3 menunjukkan perseptron, yang diperkenalkan pada bab sebelumnya, dengan bobot sinaptik terhitung sehingga dapat mewujudkan sebuah gerbang OR (verifikasi). Fokus utamanya adalah pertama untuk menangani masalah XOR dan kemudian untuk memperpanjang prosedur untuk kasus-kasus yang lebih umum dari kelas yang dipisahkan secara non-linear.



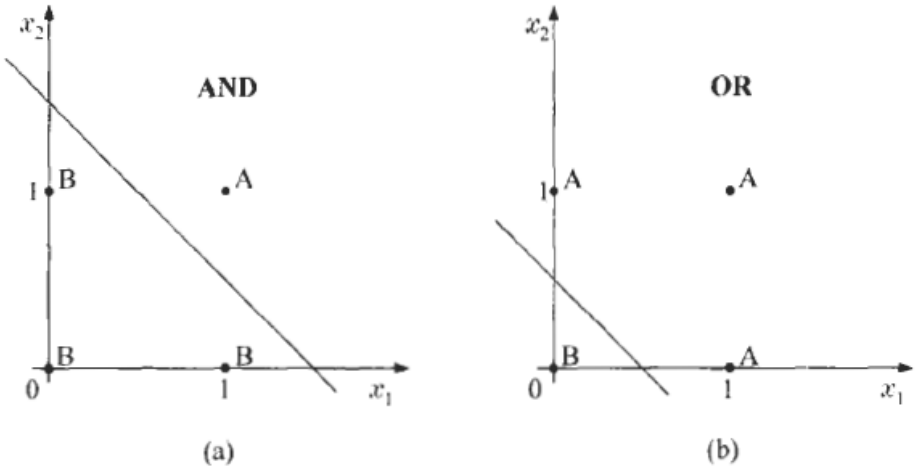
Gambar 4.1 Kelas A dan B untuk Permasalahan XOR

4.3 PERSEPTRON DUA LAPIS

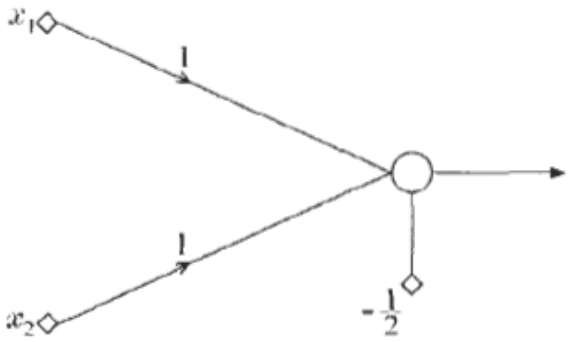
Untuk memisahkan dua kelas A dan B pada Gambar 4.1, ide pertama yang muncul dalam pikiran adalah menarik dua garis lurus, bukan satu garis lurus. Gambar 4.4 menunjukkan dua garis yang mungkin muncul yaitu g1(x) = g2(x) = 0, serta daerah di ruang g1(x) >< 0, g2(x)><0. Kelas sekarang dapat dipisahkan. Kelas A adalah ke kanan (+) dari gl (x) dan ke kiri (-) dari g2 (x). Daerah yang sesuai untuk kelas B terletak ke kiri atau ke kanan dari kedua garis tersebut. Apa yang telah dilakukan adalah untuk mengatasi masalah dalam dua tahap berturut-turut. Tahap pertama kita menghitung posisi vektor x sehubungan dengan masing-masing dua garis keputusan. Pada tahap kedua kita menggabungkan hasil pada tahap sebelumnya dan mencari posisi x terhadap kedua garis, yaitu, di luar atau di dalam area berbayang. Sekarang kita akan melihat ini dari perspektif yang sedikit berbeda, sehingga akan memudahkan kita untuk proses generalisasi.

Tabel 4.2 Tabel Kebenaran untuk Permasalahan AND dan OR

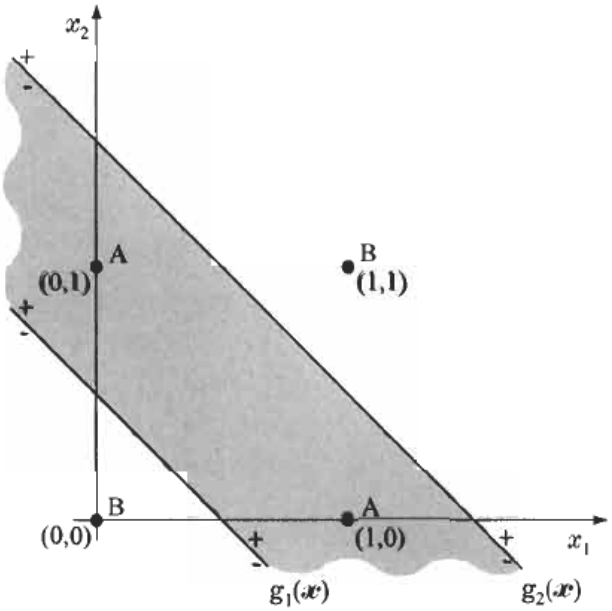
x_1	x_2	AND	Class	OR	Class
0	0	0	B	0	B
0	1	0	B	1	A
1	0	0	B	1	A
1	1	1	A	1	A



Gambar 4.2 Kelas A dan B untuk Permasalahan AND dan OR



Gambar 4.3 Perseptron dalam bentuk Gerbang OR



Gambar 4.4 Garis keputusan diwujudkan dengan perseptron dua lapis untuk permasalahan XOR.

Tabel 4.3 Tabel Kebenaran untuk dua tahap komputasi dari permasalahan XOR

1st Phase				
x_1	x_2	y_1	y_2	2nd Phase
0	0	0 (−)	0 (−)	B (0)
0	1	1 (+)	0 (−)	A (1)
1	0	1 (+)	0 (−)	A (1)
1	1	1 (+)	1 (+)	B (0)

Realisasi dari dua jalur keputusan (hyperplanes), g1 (.) dan g2 (.), selama komputasi tahap pertama dicapai dengan penerapan dua perseptron dengan input x1, x2 dan bobot sinaptik yang sesuai. Keluaran yang sesuai yaitu yi = f (gi (x)), i = 1,2, dimana fungsi aktivasi f (.) merupakan fungsi step dengan level 0 dan 1. Tabel 4.3 merangkum nilai yi untuk semua kemungkinan kombinasi dari input. Hal ini tidak lain adalah posisi relatif dari input vektor x sehubungan dengan masing-masing dua garis. Dari sudut pandang lain, perhitungan selama tahap pertama akan melakukan pemetaan dari vektor input x ke bentuk baru y = [y1, y2]^T.

Keputusan selama tahap kedua didasarkan pada data yang berubah, sebagaimana tujuan sekarang yaitu untuk memisahkan [y1, y2] = [0, 0] dan [y1, y2] = [1, 1], yang berhubungan dengan vektor kelas B, dari [y1, y2] = [1, 0], yang sesuai dengan vektor kelas A. Sebagaimana terlihat pada Gambar 4.5, dapat dengan mudah dicapai dengan menggambar garis ketiga g (y), yang dapat direalisasikan melalui neuron ketiga. Dengan kata lain, pemetaan tahap pertama mengubah masalah yang terpisah secara non-linear menjadi masalah yang terpisah secara linier. Gambar 4.6 memberikan kemungkinan realisasi langkah ini. Masing-masing tiga garis diwujudkan melalui neuron dengan bobot sinaptik yang sesuai.
Hasil arsitektur multilapis dapat dianggap sebagai sebuah generalisasi dari perseptron, dan dikenal sebagai perseptron dua lapis atau jaringan saraf dua lapis umpan maju (two-layer feed forward). Dua neuron (simpul) dari lapisan pertama melakukan perhitungan tahap pertama dan sering disebut lapisan tersembunyi. Neuron tunggal pada lapisan kedua melakukan perhitungan tahap final dan merupakan lapisan output. Pada Gambar 4.6 ditunjukkan lapisan masukan sesuai dengan (non-proses) node dimana input data dimasukkan. Dengan demikian, jumlah node input layer sama dengan dimensi ruang input. Perhatikan bahwa pada node input layer tidak terjadi pengolahan data. Garis yang ditunjukkan oleh perseptron dua lapisan pada gambar adalah:

$$g_1(\boldsymbol{x}) = x_1 + x_2 - \frac{1}{2} = 0$$

$$g_2(\boldsymbol{x}) = x_1 + x_2 - \frac{3}{2} = 0$$

$$g(\boldsymbol{y}) = y_1 - y_2 - \frac{1}{2} = 0$$

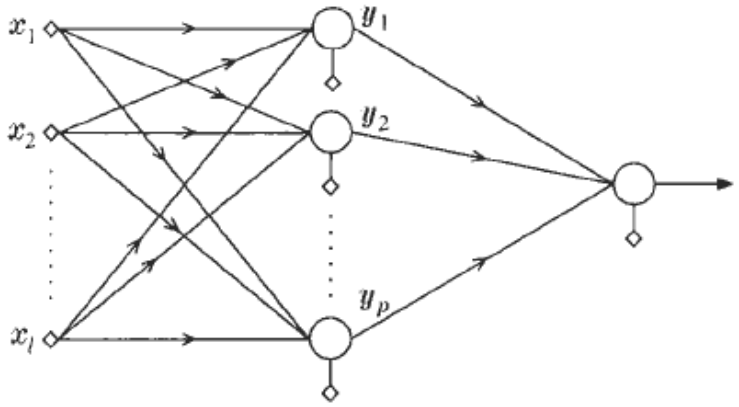
Arsitektur perseptron multilapis pada Gambar 4.6 dapat digeneralisasi untuk vektor masukan dimensi l dan untuk dua (satu) neuron atau lebih pada lapisan tersembunyi (output). Sekarang kita alihkan perhatian pada penyelidikan kemampuan diskriminatif kelas dari jaringan tersebut untuk klasifikasi tugas non linier yang lebih rumit.

4.3.1 Kemampuan Klasifikasi dari Perseptron Dua Lapis

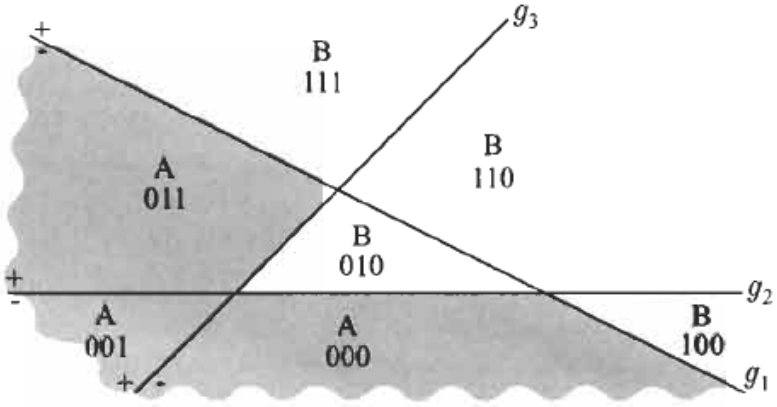
Perseptron dua lapisan pada gambar 4.6 menunjukkan bahwa aksi neuron pada lapisan tersembunyi sebenarnya merupakan pemetaan ruang input x ke titik dari sebuah persegi dengan panjang sisi unit di ruang dua-dimensi (Gambar 4.5). Untuk kasus yang lebih umum, akan dipertimbangkan vektor masukan dalam ruang dimensi l, yaitu, x $\in$ R^l, dan neuron p pada lapisan tersembunyi (Gambar 4.7). Kita akan menetapkan satu neuron output saja, meskipun hal ini juga dapat dengan mudah digeneralisasi ke banyak neuron. Menggunakan fungsi aktivasi langkah, pemetaan dari ruang input, dilakukan oleh lapisan tersembunyi, ke titik dari hiperkubus dari panjang sisi unit dalam ruang dimensi p, dilambangkan dengan H_p. Ini didefinisikan sebagai

$$H_p = \{[y_1, \ldots, y_p]^T \in \mathcal{R}^p, y_i \in [0, 1], 1 \leq i \leq p\}$$

Simpul dari hiperkubus adalah semua titik [y1,…,yp]T dari Hp dengan yi $\in$ {0,1}, 1 ≤ I ≤ p.



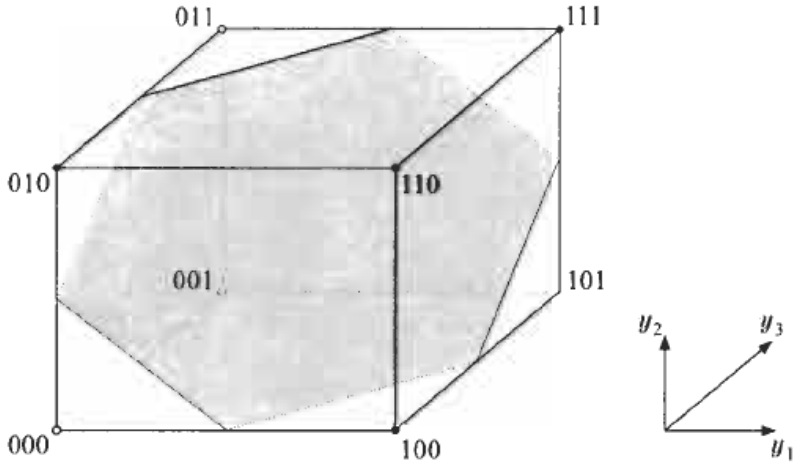
Gambar 4.7 Perseptron Dua Lapis



Gambar 4.8 Polyhedra yang dibentuk dari neuron pada layer pertama yang tersembunyi dari perseptron multilapis.

Pemetaan ruang input ke vertex dari hiperkubus dicapai melalui pembuatan hyperplane p. Setiap hyperplanes dibuat oleh neuron pada lapisan tersembunyi, dan output dari setiap neuron 0 atau 1, tergantung pada posisi yang relevan dari vektor input sehubungan dengan hyperplane terkait. Gambar 4.8 adalah contoh dari tiga persilangan hyperplane (tiga neuron) dalam ruang dimensi dua. Setiap daerah ditentukan oleh interseksi hyperplanes ini disesuaikan kepada sebuah node dalam unit hiperkubus dimensi tiga, tergantung pada posisinya sehubungan dengan masing-masing hyperplanes. Dimensi ke-i dari vertex menunjukkan posisi daerah sehubungan dengan hyperplane g_i. Sebagai contoh, vertex 001 sesuai dengan daerah yang berada pada sisi (-) dari g1, pada sisi (-) dari g2, dan pada sisi (+)

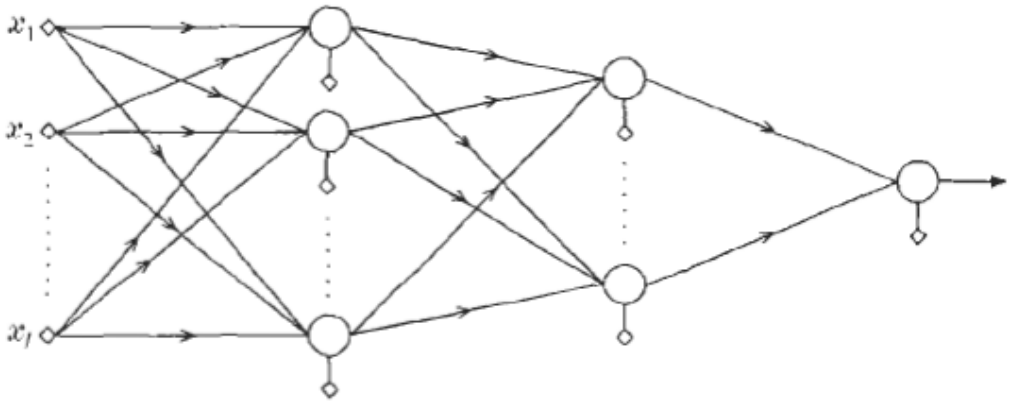
dari g3. Jadi, kesimpulan yang dapat diambil adalah bahwa lapisan pertama neuron membagi input ruang dimensi-1 ke polyhedra yang dibentuk dari interseksi hyperplane. Semua vektor yang terletak dalam satu polyhedral daerah ini dipetakan ke sebuah titik tertentu dari unit hiperkubus Hp. Pada saat neuron output menyadari keberadaan hyperplane lain, yang memisahkan hiperkubus menjadi dua bagian, memiliki beberapa titik pada satu sisi dan beberapa titik di sisi lain. Neuron ini menyediakan perseptron multilapis dengan kemampuan untuk mengklasifikasi vektor ke dalam kelas yang terdiri dari gabungan dari daerah polyhedral. Mari kita pertimbangkan misalnya bahwa kelas A terdiri dari gabungan dari daerah-daerah yang dipetakan ke titik 000, 001, 011 dan kelas B terdiri dari sisanya (Gambar 4.8). Gambar 4.9 menunjukkan unit H_3 (hiper) kubus dan unit (hiper) plane yang memisahkan ruang R^3 menjadi dua daerah dengan (kelas A), titik 000, 001, 011 di satu sisi dan (kelas B) vertex 010, 100, 110, 111 di sisi lain. Inilah plane-nya yaitu -y1 - y2 + y3 + 0,5 = 0, yang diwujudkan oleh neuron output. Dengan menggunakan konfigurasi, semua vektor dari kelas A menghasilkan sebuah output 1 (+) dan semua vektor dari kelas B di output 0 (-). Di sisi lain, jika kelas A terdiri dari gabungan 000 U 111 U 110 dan sisi lain yaitu kelas B adalah sisanya, maka tidak mungkin untuk membangun sebuah plane yang memisahkan kelas A dari vertex kelas B. Dengan demikian, dapat disimpulkan bahwa suatu perseptron dua lapis dapat memisahkan masing-masing kelas yang terdiri dari gabungan beberapa daerah polyhedral tetapi bukan gabungan dari setiap kesatuan daerah tersebut. Semua tergantung pada posisi relatif dari vertex dari Hp, dimana kelas dipetakan, dan apakah ini secara linear dapat dipisahkan atau tidak. Sebelum kita melangkah lebih jauh untuk melihat cara untuk mengatasi kelemahan ini, harus dipahami bahwa 101 titik kubus tidak sesuai dengan salah satu daerah polyhedral. Beberapa vertex tersebut dikatakan menyesuaikan dengan polyhedra virtual, dan mereka tidak mempengaruhi tugas klasifikasi.



Gambar 4.9 Neuron dari layer pertama yang tersembunyi yang memetakan vektor input ke dalam satu titik unit (hyper) kubus. Neuron output menyadari sebuah (hyper) plane ke titik terpisah menurut label kelas mereka.

4.4 PERSEPTRON TIGA LAPIS

Ketidakmampuan perseptron dua lapis untuk memisahkan kelas yang dihasilkan dari beberapa gabungan daerah polyhedral dan dari fakta bahwa neuron output hanya dapat mewujudkan hyperplane tunggal. Ini adalah situasi yang sama dari perseptron dasar saat dihadapkan dengan masalah XOR. Kesulitan ini diatasi dengan membuat dua garis, bukan satu. Disini mengadopsi jalur yang sama untuk melarikan diri.



Gambar 4.10 Arsitektur dari perseptron multilapis dengan dua lapis neuron tersembunyi dan neuron output tunggal

Gambar 4.10 menunjukkan arsitektur perseptron tiga lapis dengan dua lapis neuron tersembunyi dan satu lapis output. Disini akan ditunjukkan, secara konstruktif, bahwa sebagaimana arsitektur dapat memisahkan kelas yang dihasilkan dari beberapa gabungan daerah polyhedral. Kita asumsikan bahwa semua daerah kepentingan dibentuk oleh interseksi setengah-ruang dimensi-l yang didefinisikan oleh hyperplane p. Hal ini direalisasikan dengan p neuron pada lapisan tersembunyi pertama, yang juga melakukan pemetaan input ruang ke titik Hp hiperkubus dari panjang sisi unit. Dalam sekuel ini marilah kita asumsikan bahwa kelas A terdiri dari gabungan J yang dihasilkan dari polyhedra, dan kelas B adalah sisanya. Kemudian kita menggunakan J neuron pada lapisan tersembunyi kedua. Masing-masing neuron menyadari sebuah hyperplane dalam ruang dimensi p. Bobot sinaptik untuk masing-masing neuron lapisan kedua dipilih sehingga hyperplane tersebut hanya meninggalkan satu titik Hp pada satu sisi dan semua titik sisanya di sisi yang lain. Untuk setiap neuron dengan titik berbeda yang terisolasi, yaitu salah satu vertex J kelas A. Dengan kata lain, setiap waktu suatu vektor input dari kelas A memasuki jaringan, salah satu neuron J hasil lapisan kedua menghasilkan nilai 1 dan sisanya J - 1 memberikan nilai 0. Sebaliknya, untuk vektor kelas B semua neuron di output lapis kedua adalah 0. Klasifikasi menjadi tugas yang benar-benar harus dilakukan. Neuron lapisan keluaran dipilih untuk mewujudkan sebuah gerbang OR. Outputnya akan bernilai 1 untuk kelas A dan 0 untuk vektor kelas B.

Jumlah neuron pada lapisan tersembunyi kedua dapat dikurangi dengan memanfaatkan geometri yang dihasilkan dari setiap masalah spesifik, misalnya, pada saat dua dari vertex J ditempatkan pada jalur yang membuat mereka terpisah dari yang lain, menggunakan sebuah hyperplane tunggal. Akhirnya, struktur multilapis dapat digeneralisasi untuk lebih dari dua kelas. Untuk tujuan ini, neuron lapisan output harus dinaikkan jumlahnya, dengan membuat satu gerbang OR untuk tiap kelasnya. Oleh karena itu, salah satu dari mereka menghasilkan nilai 1 setiap kali vektor dari masing-masing kelas masuk ke jaringan, dan lainnya memberikan nilai 0. Jumlah neuron lapisan kedua juga akan terpengaruh (mengapa?).

Singkatnya, kita dapat mengatakan bahwa neuron lapisan tersembunyi pertama membentuk hyperplane, lapisan kedua membentuk region, dan akhirnya neuron lapisan output membentuk kelas.

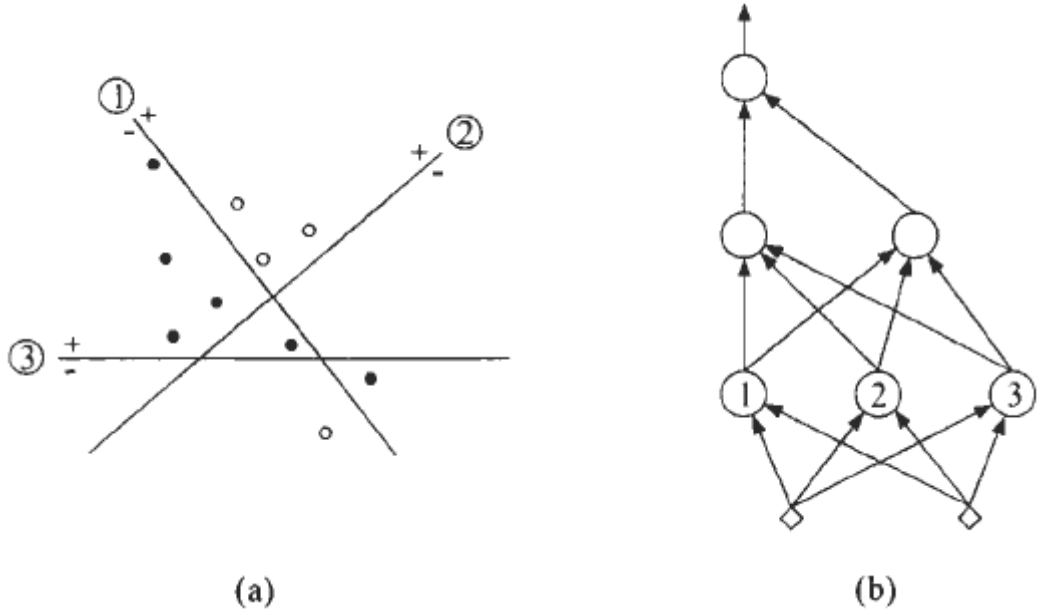
Sejauh ini, kita telah difokuskan pada kemampuan potensial dari perseptron lapis tiga untuk memisahkan setiap gabungan daerah polyhedral. Untuk mengasumsikan bahwa pada prakteknya kita tahu daerah dimana data berada dan kita dapat menghitung setiap persamaan hyperplane secara analitis dan hal itu tidak diragukan lagi, hanya saja ini menjadi tujuan yang

belum terealisasi. Kita tahu bahwa dalam prakteknya dititik-beratkan pada daerah tersebut. Seperti halnya dengan kasus perseptron ini, seseorang harus memakai algoritma pembelajaran yang mempelajari bobot sinaptik dari vektor data pelatihan yang tersedia. Ada dua arah utama dimana kita akan fokuskan perhatian kita. Salah satu arahnya adalah jaringan dibangun dengan cara yang benar untuk mengklasifikasikan semua data pelatihan yang tersedia, dengan membangunnya sebagai serangkaian pengklasifikasian linear. Arah lain yaitu mengurangi dirinya dari kendala klasifikasi yang benar dan menghitung bobot sinaptik sehingga dapat meminimalkan sebuah fungsi biaya yang dipilih.

4.5 ALGORITMA BERDASARKAN KLASIFIKASI EKSAK DARI SET PELATIHAN

Titik awal dari teknik ini adalah arsitektur kecil (biasanya tidak mampu mengatasi masalah), yang ditambah berturut-turut sampai klasifikasi yang benar dari semua vektor fitur N dari set pelatihan X tercapai. Algoritma yang berbeda mengikuti cara yang berbeda untuk meningkatkan arsitektur mereka. Dengan demikian, beberapa algoritma memperluas arsitektur mereka dalam hal jumlah lapisan [Meza 89, Frea 90], sementara yang lain menggunakan satu atau dua lapisan tersembunyi dan memperluas dalam hal jumlah node (neuron) [Kout 94, Bose 96]. Selain itu, beberapa algoritma [Frea 90] memungkinkan koneksi antara node dari lapisan *nonsuccessive*. Sedangkan yang lainnya memungkinkan koneksi antara node dari lapisan yang sama [Refe 91]. Prinsip umum yang diadopsi oleh sebagian besar teknik ini adalah dekomposisi masalah menjadi masalah yang lebih kecil yang lebih mudah ditangani. Untuk setiap masalah yang lebih kecil, digunakan node tunggal. Parameter-parameternya ditentukan baik secara iterasi menggunakan algoritma belajar yang tepat, seperti algoritma saku atau algoritma LMS (Bab 3), maupun secara langsung melalui perhitungan analitis. Dari cara algoritma ini dalam membangun jaringan, maka sering disebut sebagai teknik konstruktif.

Algoritma *tiling* [Meza 89] menyusun arsitektur dengan banyak lapisan (biasanya lebih dari tiga lapis). Kami menggambarkan algoritma untuk kasus dua kelas (A dan B).



Gambar 4.11 Garis Keputusan dan arsitektur sesuai hasil algoritma *tiling*. Lingkaran terbuka disesuaikan dengan kelas A (B).

Algoritma ini dimulai dengan node tunggal, n(X), pada lapisan pertama, yang disebut unit master dari lapisan ini.

Node ini dilatih dengan menggunakan algoritma *pocket* (Bab 3) dan setelah menyelesaikan pelatihan, algoritma ini membagi set data pelatihan X menjadi dua subset X^+ dan X^- (baris 1 pada Gambar 4.11). Jika X^+ (X^-) berisi fitur vektor dari kedua kelas, kita menambahkan sebuah node tambahan, $n(X^+)$ ($n(X^-)$), yang disebut unit tambahan. Node ini dilatih hanya menggunakan vektor fitur di X^+ (X^-) (baris 2). Jika salah satu dari X^{++}, X^{+-} (X^{-+}, X^{--}) yang dihasilkan oleh neuron $n(X^+)$ ($n(X^-)$) berisi vektor-vektor dari kedua kelas tersebut, ditambahkan node tambahan lainnya. Prosedur ini berhenti setelah sejumlah langkah tertentu, karena vektor unit baru yang ditambahkan (tambahan) harus membedakan penurunan di setiap langkah. Jadi, lapisan pertama terdiri dari satu unit master tunggal dan, secara umum, lebih dari satu unit tambahan. Mudah untuk menunjukkan bahwa cara ini berhasil, sehingga tidak ada dua vektor yang berasal dari kelas yang berbeda yang memberikan output lapisan pertama yang sama.

Misal X1 = {y : y = f1 (x), x Є X), di mana f1 adalah pemetaan dilakukan oleh lapisan pertama. Penerapan prosedur yang baru saja dijelaskan ke set X1 dari sampel y yang berubah, kita dapat membangun lapisan kedua dari arsitektur dan sebagainya. Dalam [Meza 89] ditunjukkan bahwa pilihan yang cocok dari bobot-bobot antara dua lapisan yang berdekatan memastikan bahwa setiap unit master yang baru ditambahkan dapat mengklasifikasikan semua vektor terklasifikasi dengan benar oleh unit master dari lapisan sebelumnya, ditambah sedikitnya satu vektor lagi. Dengan demikian, algoritma *tiling* menghasilkan arsitektur yang mengklasifikasikan dengan benar semua pola X di sejumlah langkah tertentu.

Pengamatan yang menarik adalah bahwa semua kecuali lapisan pertama dalam memperlakukan vektor biner. Hal ini mengingatkan kita pada unit hiperkubus dari bagian sebelumnya. Memobilisasi argumen yang sama seperti sebelumnya, kita dapat menunjukkan bahwa algoritma ini dapat memudahkan dalam mengkoreksi klasifikasi arsitektur yang memiliki paling banyak tiga lapisan node. Algoritma konstruktif lainnya dibangun berdasarkan gagasan aturan klasifikasi tetangga terdekat, yang telah dibahas dalam Bab 2. Neuron dari lapisan pertama melaksanakan hyperplanes yang membelah segmen garis yang bergabung dengan vektor fitur pelatihan [Murp 90]. Lapisan kedua membentuk daerah, dengan menggunakan sejumlah neuron yang sesuai untuk menerapkan gerbang AND, dan kelas-kelas yang terbentuk melalui neuron lapisan akhir, yang mengimplementasikan gerbang OR. Kelemahan utama teknik ini adalah keterlibatan neuron dalam jumlah yang sangat besar. Teknik yang digunakan untuk mengurangi jumlah ini juga telah diusulkan ([Kout 94, Bose 96]).

4.6 ALGORITMA PROPAGASI BALIK (*BACKPROPAGATION ALGORITHM*)

Arah lain yang akan diikuti untuk merancang perseptron multilapis adalah memperbaiki arsitektur dan menghitung parameter sinaptiknya sehingga dapat meminimalkan fungsi biaya yang tepat dari outputnya. Ini adalah pendekatan yang paling populer sejauh ini, yang tidak hanya mengatasi kekurangan dari jaringan besar yang dihasilkan oleh bagian sebelumnya tetapi juga membuat jaringan yang kuat untuk sejumlah aplikasi lainnya, di luar pengenalan pola. Namun, pendekatan semacam ini akan segera dihadapkan dengan kesulitan serius. Seperti misalnya diskontinuitas dari fungsi (aktivasi) langkah, melarang diferensiasi terhadap parameter yang tidak diketahui (bobot sinaptik). Diferensiasi dilakukan sebagai akibat dari prosedur minimisasi fungsi biaya. Dalam hal ini kita akan melihat bagaimana kesulitan ini dapat diatasi.

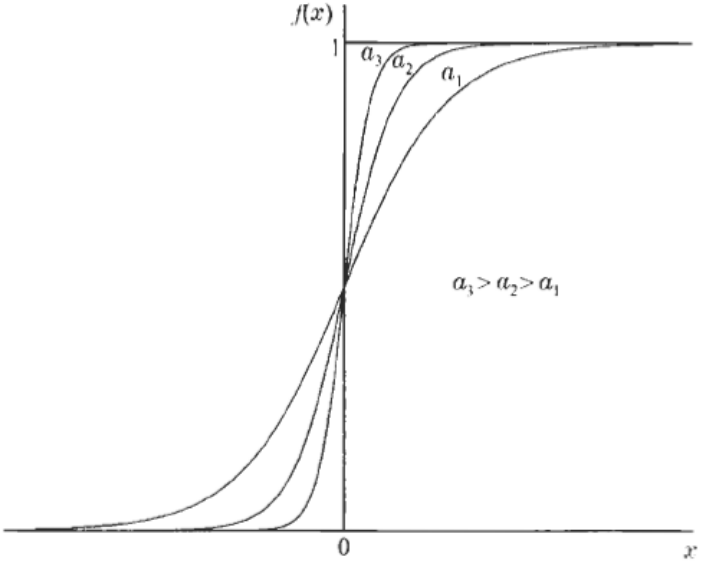
Arsitektur perseptron multilapis yang kita diskusikan sejauh ini telah dikembangkan di sekitar neuron McCulloch-Pitts, yang menggunakan fungsi aktivasi fungsi step:

$$f(x) = \begin{cases} 1 & x > 0 \\ 0 & x < 0 \end{cases}$$

Fungsi diferensial kontinyu yang sudah dikenal, yang merupakan pendekatan dari fungsi step, adalah keluarga dari fungsi sigmoid. Direpresentasikan dengan fungsi logistik

$$f(x) = \frac{1}{1 + \exp(-ax)} \qquad (4.1)$$

di mana a adalah parameter *slope*.
Gambar 4.12 menunjukkan fungsi sigmoid untuk nilai a yang berbeda-beda, dalam suatu fungsi step. Kadang-kadang variasi a dari fungsi logistik digunakan dan bersifat *antisymmetric* yang berkaitan dengan fungsi asal, yaitu, f(-x) = - f(x).



Gambar 4.12 Fungsi Logistik

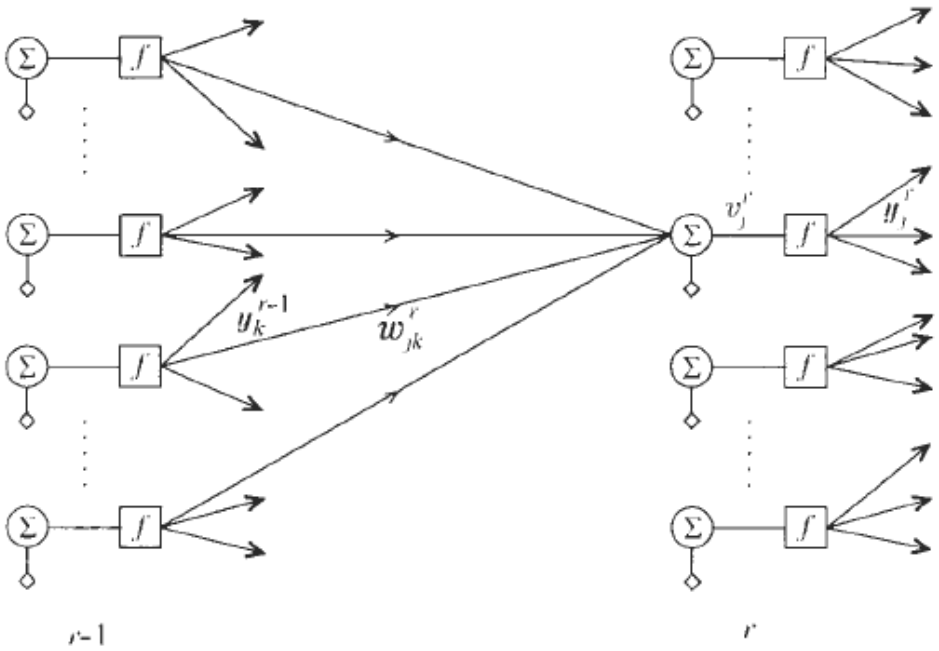
Didefinisikan sebagai:

$$f(x) = \frac{2}{1 + \exp(-ax)} - 1 \qquad (4.2)$$

Nilai ini bervariasi antara 1 dan -1, dan merupakan keluarga fungsi tangen hiperbolik

$$f(x) = c\frac{1 - \exp(-ax)}{1 + \exp(-ax)} = c\tanh\left(\frac{ax}{2}\right) \qquad (4.3)$$

Semua fungsi-fungsi ini juga dikenal sebagai fungsi *squashing* karena output mereka terbatas dalam rentang nilai yang terbatas. Dalam rangkaian ini, kita akan mengadopsi arsitektur syaraf multilapis seperti pada Gambar 4.10, dan kita akan mengasumsikan bahwa fungsi aktivasinya seperti bentuk yang diberikan dalam persamaan (4.1) - (4.3). Tujuannya adalah untuk menurunkan algoritma pelatihan berulang yang menghitung bobot sinaptik jaringan sehingga fungsi biaya yang sesuai pilihan dapat diminimalkan. Sebelum masuk ke derivasi dari sebuah skema, sesuatu yang penting harus diklarifikasi terlebih dahulu. Saat kita menjauh dari fungsi step, semua yang dikatakan sebelumnya tentang pemetaan input vektor ke titik dari unit hiperkubus tidak berlaku lagi. Sekarang, fungsi biaya yang mengambil beban untuk klasifikasi yang benar.



Gambar 4.13 Definisi variabel yang terlibat dalam algoritma propagasi balik

Untuk alasan generalisasi, mari kita asumsikan bahwa jaringan terdiri dari sejumlah L lapisan neuron yang tetap, dengan node k_0 pada lapisan input dan neuron k_r pada lapisan ke-r, untuk r = 1,2,. . . , L. Jelasnya, k_0 sama dengan l. Semua neuron menggunakan fungsi aktivasi sigmoid yang sama. Seperti halnya dalam Bagian 3.3, kita asumsikan bahwa ada N pasang pelatihan yang tersedia (y(i), x(i)), i = 1, 2,. . . , N. Karena sekarang kita mengasumsikan neuron output k_L, output tidak lagi skalar tetapi vektor berdimensi k_L, y(i) = [y1 (i),. . . , Yk_L (i)]^T. Input (fitur) vektor merupakan vektor berdimensi k_0, x(i) = [x1 (i),. . . , xk_0 (i)]^T. Selama pelatihan, pada saat vektor x(i) diterapkan ke input, output dari jaringan akan menjadi ŷ(i), yang berbeda dari nilai yang diharapkan, y(i). Bobot sinaptik dihitung supaya sebuah fungsi biaya J yang sesuai (untuk setiap masalah), yang tergantung pada nilai-nilai y(i) dan ŷ(i), i = I, 2,. . . , N, adalah minimal. Jelas bahwa J bergantung, melalui ŷ(i), pada bobot dan bahwa ini adalah kebergantungan yang nonlinier, karena sifat dari jaringan itu sendiri. Jadi, minimisasi fungsi biaya dapat dicapai melalui teknik iteratif. Pada bagian ini kita akan menerapkan skema *gradient descent* (Lampiran C), yang merupakan pendekatan yang paling banyak digunakan. Biarkan w^r_j menjadi vektor bobot (termasuk ambang batas) dari neuron ke-j pada lapisan ke-r, yang merupakan vektor berdimensi $k_{r-1} + 1$ dan didefinisikan sebagai (Gambar 4.13) $w^r_j = [w^r_{j0}, w^r_{j1}, \ldots, w^r_{jkr-1}]^T$. Langkah iterasi dasar akan berbentuk:

$$\boldsymbol{w}_j^r(\text{new}) = \boldsymbol{w}_j^r(\text{old}) + \Delta \boldsymbol{w}_j^r$$

Dengan

$$\Delta \boldsymbol{w}_j^r = -\mu \frac{\partial J}{\partial \boldsymbol{w}_j^r} \qquad (4.4)$$

dimana w^r_j (lama) adalah perkiraan saat ini dari bobot yang tidak diketahui dan Δw^r_j faktor koreksi yang sesuai untuk mendapatkan w^r_j perkiraan selanjutnya (baru).

Pada Gambar 4.13 v^r_j adalah penjumlahan berbobot dari input ke neuron ke-j dari lapisan ke-r dan y^r_j merupakan output yang sesuai setelah fungsi aktivasi. Dalam rangkaian ini kita akan memusatkan perhatian pada fungsi biaya dalam bentuk:

$$J = \sum_{i=1}^{N} \mathcal{E}(i) \qquad (4.5)$$

dimana ε adalah fungsi yang didefinisikan secara tepat yang bergantung pada ŷ(i) dan y(i), i = 1, 2, . . , N. Dengan kata lain, J dinyatakan sebagai jumlah dari nilai-nilai N yang fungsi ε mengambil untuk setiap pasangan pelatihan (y(i), x(i)). Sebagai contoh, kita dapat memilih ε(i) sebagai jumlah dari kuadrat kesalahan dari neuron output.

$$\mathcal{E}(i) = \frac{1}{2} \sum_{m=1}^{k_L} e_m^2(i) \equiv \frac{1}{2} \sum_{m=1}^{k_L} (y_m(i) - \hat{y}_m(i))^2, \quad i = 1, 2, \ldots, N \qquad (4.6)$$

Untuk perhitungan dari bagian koreksi pada persamaan (4.4) gradien dari fungsi biaya J dihubungkan dengan bobot yang diperlukan dan juga evaluasi dari $\delta\varepsilon(i)/\delta w^r_j$.

Perhitungan Gradien

Biarkan $y^{r-1}_k(i)$ menjadi output dari neuron ke-k, k = 1,2,. . . , k_{r-1}, pada lapisan ke-(r-1) untuk pasangan pelatihan ke-i dan w^r_{jk} perkiraan saat ini dari bobot yang sesuai menuju ke neuron ke-j pada lapisan ke-r, dengan j = 1,2,. . . ,k_r (Gambar 4.13). Dengan demikian, argumen dari fungsi aktivasi f (.) dari neuron terakhir akan menjadi:

$$v^r_j(i) = \sum_{k=1}^{k_{r-1}} w^r_{jk} y^{r-1}_k(i) + w^r_{jo} \equiv \sum_{k=0}^{k_{r-1}} w^r_{jk} y^{r-1}_k(i) \qquad (4.7)$$

Dimana berdasarkan definisi $y^r_0(i) = +1$, Vr, 1; termasuk ambang batas dalam bobot. Untuk lapisan output, kita mempunyai r = L. $y^r_k(i) = \hat{y}_k(i)$, k=1,2,…,k_L, yang merupakan output dari jaringan syaraf, dan untuk r = 1, $y^{r-1}_k(i) = x_k(i)$, k=1,2,…,k_0 yang merupakan input jaringan.
Sebagaimana yang ditunjukkan dari persamaan (4.7), kebergantungan dari ε(i) pada w^r_j telah melampaui $v^r_j(i)$. Dengan aturan rantai dalam diferensiasi, kita memiliki:

$$\frac{\partial \mathcal{E}(i)}{\partial \boldsymbol{w}^r_j} = \frac{\partial \mathcal{E}(i)}{\partial v^r_j(i)} \frac{\partial v^r_j(i)}{\partial \boldsymbol{w}^r_j} \qquad (4.8)$$

Dari persamaan 4.7 diperoleh:

$$\frac{\partial}{\partial \boldsymbol{w}^r_j} v^r_j(i) \equiv \begin{bmatrix} \frac{\partial}{\partial w^r_{j0}} v^r_j(i) \\ \vdots \\ \frac{\partial}{\partial w^r_{jk_{r-1}}} v^r_j(i) \end{bmatrix} = \boldsymbol{y}^{r-1}(i) \qquad (4.9)$$

dimana

$$\boldsymbol{y}^{r-1}(i) = \begin{bmatrix} +1 \\ y^{r-1}_1(i) \\ \vdots \\ y^{r-1}_{k_{r-1}}(i) \end{bmatrix} \qquad (4.10)$$

Kita mendefinisikan

$$\frac{\partial \mathcal{E}(i)}{\partial v_j^r(i)} \equiv \delta_j^r(i) \qquad (4.11)$$

Sehingga persamaan 4.4 menjadi:

$$\Delta \boldsymbol{w}_j^r = -\mu \sum_{i=1}^{N} \delta_j^r(i) \boldsymbol{y}^{r-1}(i) \qquad (4.12)$$

Hubungan (4.12) adalah umum untuk setiap fungsi biaya yang terdiferensiasi dalam bentuk persamaan (4,5). Dalam rangkaian ini kita akan menghitung nilai δ^r_j (i) untuk kasus khusus dari kuadrat terkecil (4.6). Prosedurnya sama untuk pemilihan fungsi biaya alternatif.

Perhitungan δ^r_j (i) untuk Fungsi Biaya pada persamaan (4.6)

Perhitungan dimulai dari r = L dan propagasi balik untuk r = L - 1, L - 2,. . . , 1. Inilah sebabnya mengapa algoritma yang akan diturunkan ini dikenal sebagai algoritma propagasi balik.

(i) r = L

$$\delta_j^L(i) = \frac{\partial \mathcal{E}(i)}{\partial v_j^L(i)} \qquad (4.13)$$

$$\mathcal{E}(i) \equiv \frac{1}{2} \sum_{m=1}^{k_L} e_m^2(i) \equiv \frac{1}{2} \sum_{m=1}^{k_L} (f(v_m^L(i)) - y_m(i))^2 \qquad (4.14)$$

Sehingga:

$$\delta_j^L(i) = e_j(i) f'(v_j^L(i)) \qquad (4.15)$$

dimana f'adalah turunan dari f (.). Pada lapisan terakhir, ketergantungan ε (i) pada v^r_j adalah eksplisit dan perhitungan derivatif adalah langsung. Hal ini sebenarnya tidak dibenarkan, namun diperbolehkan untuk lapisan tersembunyi, dimana perhitungan dari penurunan memerlukan elaborasi yang lebih.

(ii) r < L

Karena adanya kebergantungan yang berturut-turut antar lapisan, nilai $v^{r-1}{}_j(i)$ mempengaruhi seluruh nilai $v^r_k(i)$ VL, k = 1, 2, .., k_r, dari lapisan berikutnya. Dengan menggunakan aturan rantai dalam diferensiasi sekali lagi, kita memperoleh:

$$\frac{\partial \mathcal{E}(i)}{\partial v_j^{r-1}(i)} = \sum_{k=1}^{k_r} \frac{\partial \mathcal{E}(i)}{\partial v_k^r(i)} \frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)} \qquad (4.16)$$

dan dari masing-masing definisi (4.11)

$$\delta_j^{r-1}(i) = \sum_{k=1}^{k_r} \delta_k^r(i) \frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)} \qquad (4.17)$$

Tapi

$$\frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)} = \frac{\partial \left[\sum_{m=0}^{k_{r-1}} w_{km}^r y_m^{r-1}(i) \right]}{\partial v_j^{r-1}(i)} \qquad (4.18)$$

dengan

$$y_m^{r-1}(i) = f(v_m^{r-1}(i)) \qquad (4.19)$$

Sehingga

$$\frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)} = w_{kj}^r f'(v_j^{r-1}(i)) \qquad (4.20)$$

Dari persamaan (4.10) dan (4.11) diperoleh hasil berikut:

$$\delta_j^{r-1}(i) = \left[\sum_{k=1}^{k_r} \delta_k^r(i) w_{kj}^r\right] f'(v_j^{r-1}(i)) \qquad (4.21)$$

dan untuk keseragaman dengan persamaan (4.15)

$$\delta_j^{r-1}(i) = e_j^{r-1}(i) f'(v_j^{r-1}(i)) \qquad (4.22)$$

dimana

$$e_j^{r-1}(i) = \sum_{k=1}^{k_r} \delta_k^r(i) w_{kj}^r \qquad (4.23)$$

Hubungan (4.15), (4,22), (4,23) merupakan iterasi yang mengarah ke perhitungan δ^r_j, r = 1, 2,…, L, j = 1,2,. . . , k_r. Satu-satunya kuantitas yang belum dihitung adalah f'(.). Untuk fungsi dalam persamaan (4.1) kita memperoleh:

$$f'(x) = \alpha f(x)(1 - f(x))$$

Algoritma sekarang telah diturunkan. Skema algoritma pertama kali disajikan dalam [Werb 74] dalam formulasi yang lebih umum.

Algoritma Propagasi Balik

- Inisialisasi: Inisialisasi semua bobot dengan nilai acak kecil dari pseudorandom generator urutan.
- 1. Komputasi maju: Untuk setiap vektor x (i) fitur latihan, i=1,2,…N, hitung semua v^r_j (i), y^r_j (i) = f (v^r_j (i)), j = 1,2,…k_r, r = 1,2,…L dari persamaan (4.7). Hitung fungsi biaya perkiraan bobot saat ini dari persamaan (4.5) dan (4.14).
- 2. Komputasi balik: Untuk setiap i = 1,2,…N dan j = 1,2,…kL, hitung δ^L_j (i) dari persamaan (4.15) dan kemudian hitung δ^{r-1}_j (i) dari persamaan (4.22) dan (4.23) untuk r = L, L-1,…,2, dan j = 1,2,…k_r.
- Perbarui bobot: Untuk r = 1,2,…L dan j = 1,2,…k_r

$$w_j^r(\text{new}) = w_j^r(\text{old}) + \Delta w_j^r$$

$$\Delta w_j^r = -\mu \sum_{i=1}^{N} \delta_j^r(i) y^{r-1}(i)$$

Keterangan

- Sejumlah kriteria telah diusulkan untuk mengakhiri iterasi. Dalam [Kram 89] disarankan bahwa kita mengakhiri iterasi baik ketika fungsi biaya J menjadi lebih kecil dari batas tertentu atau bila gradien yang sehubungan dengan bobot menjadi kecil. Tentu saja, yang terakhir berpengaruh langsung pada laju perubahan bobot antara langkah-langkah iterasi yang berurutan.

- Sebagaimana dengan keseluruhan algoritma yang muncul dari metode *gradient descent*, kecepatan konvergensi dari skema progagasi balik tergantung pada nilai dari pembelajaran konstan μ. Nilainya harus cukup kecil untuk menjamin konvergensi tetapi jangan terlalu kecil, karena kecepatan konvergensi akan menjadi sangat lambat. Pilihan terbaik dari nilai μ sangat tergantung pada masalah dan kondisi fungsi biaya dalam ruang bobot. Nilai minima yang lebar akan menghasilkan gradien kecil; sehingga besar nilai μ akan mengakibatkan konvergensi lebih cepat. Di sisi lain, untuk nilai-nilai minima yang curam dan sempit akan memperkecil nilai μ yang diperlukan untuk menghindari melampaui nilai minimum. Seperti yang akan segera kita lihat, skenario dengan μ adaptif juga mungkin dilakukan.
- Meminimalkan fungsi biaya untuk perseptron multilapis adalah tugas minimisasi nonlinier. Dengan demikian, adanya *local minima* (lokal minimum) yang sesuai pada permukaan fungsi biaya merupakan realitas yang diharapkan. Oleh karena itu, algoritma propagasi balik menjalankan risiko terjebak dalam sebuah lokal minimum. Jika lokal minimum cukup dalam, ini masih mungkin menjadi solusi yang baik. Namun, dalam kasus-kasus di mana hal ini tidak dibenarkan, terjebak dalam sebuah lokal minimum adalah situasi tidak diinginkan dan algoritma harus diinisialisasi ulang dengan set yang berbeda dari kondisi awal.
- Algoritma yang dijelaskan dalam bagian ini yang memperbarui bobot sekali untuk seluruh pasangan pelatihan (input-output yang diinginkan), telah muncul dalam jaringan. Modus operasi ini dikenal sebagai modus *batch*. Sebuah variasi dari pendekatan ini digunakan untuk memperbarui bobot pada setiap pasangan pelatihan. Hal ini dikenal sebagai modus pola atau modus *on-line*. Hal ini sejalan dengan LMS, dimana, menurut pendekatan Robbins-Monro, nilai sesaat dari gradien adalah dihitung dan bukan merupakan nilai *mean* (rata tengah). Dalam kasus propagasi balik, jumlah dari $\delta^r_j(i)$ untuk keseluruhan i diganti dengan masing-masing nilainya. Algoritma dalam modus pola operasinya kemudian menjadi:

$$\boldsymbol{w}_j^r(i+1) = \boldsymbol{w}_j^r(i) - \mu\delta_j^r(i)\boldsymbol{y}^{r-1}(i)$$

  Dibandingkan dengan mode pola, modus *batch* merupakan proses rata-rata yang melekat. Hal ini menyebabkan perkiraan gradien yang lebih baik, sehingga lebih memperlihatkan konvergensi berkelakuan-baik. Di sisi lain, modus pola menyajikan derajat yang lebih tinggi dari tingkat keacakan selama pelatihan. Ini mungkin membantu algoritma untuk menghindari terjebak dalam lokal minimum. Dalam [Siet 91] disarankan bahwa efek menguntungkan bahwa keacakan yang mungkin terjadi pada pelatihan dapat lebih ditekankan dengan menambahkan urutan *white noise* (kecil) dalam data pelatihan.

  Praktek lain yang biasa digunakan fokus pada cara data pelatihan yang disajikan dalam jaringan. *Selama pelatihan, pelatihan vektor yang tersedia digunakan dalam persamaan update lebih dari sekali* sampai algoritma menyatu. Satu presentasi lengkap dari semua pasangan pelatihan *N* merupakan sebuah *epoch*. Seperti *epochs* berturut-turut diterapkan, hal ini adalah praktek yang baik dari konvergensi sudut pandang untuk mengacak urutan penyajian pasangan pelatihan. Pengacakan dapat lagi membantu algoritma mode pola untuk melompat keluar dari daerah sekitar minimum lokal, saat ini terjadi. Namun, pilihan akhir antara mode batch dan pola dari operasi tergantung pada masalah spesifik [Hert 91, halaman 119].

- Sekali pelatihan jaringan telah tercapai, nilai-nilai yang mana sinapsis dan ambang telah berkumpul dibekukan dan jaringan tersebut siap untuk klasifikasi. Ini adalah tugas yang jauh lebih mudah daripada pelatihan. Tidak diketahui fitur vektor disajikan dalam input

dan diklasifikasikan di kelas yang ditunjukkan oleh output dari jaringan. Perhitungan dilakukan oleh neuron adalah dari tipe memperbanyak-menambah diikuti oleh sebuah nonlinier. Ini menyebabkan implementasi berbagai hardware mulai dari optik sampai desain chip VLSI. Selain itu, jaringan saraf memiliki sebuah sifat paralel secara natural terpasang tetap dan perhitungan dalam setiap lapisan dapat dilakukan secara paralel. Ini nyata karakteristik dari jaringan saraf telah menyebabkan perkembangan khusus *neurocomputers*, dan sejumlah mereka yang sudah tersedia secara komersial; lihat, sebagai contoh, [Koli 97].

4.7 VARIASI PADA PROPAGASI BALIK

Kedua versi dari skema *backpropagation (propagsi balik)*, mode batch dan pola, mewarisi kelemahan dari semua metode dibangun di atas pendekatan *turunan gradien*: *konvergensi mereka untuk fungsi biaya minimum lambat.* Lampiran C membahas fakta bahwa sifat ini menjadi lebih menonjol jika nilai eigen dari yang sesuai matriks Hessian menunjukkan penyebaran besar. Dalam kasus tersebut, perubahan gradien fungsi biaya antara langkah iterasi yang berurutan tidak lancar tapi berosilasi, mendahului untuk memperlambat konvergensi. Salah satu cara untuk mengatasi masalah ini adalah dengan menggunakan sebuh *batas momentum* yang menghaluskan keluaran osilasi dan mempercepat konvergensi. Algoritma propagasi balik dengan batas momentum mengambil bentuk

$$\Delta \boldsymbol{w}_j^r(\text{new}) = \alpha \Delta \boldsymbol{w}_j^r(\text{old}) - \mu \sum_i^N \delta_j^r(i) \boldsymbol{y}^{r-1}(i) \qquad (4.24)$$

$$\boldsymbol{w}_j^r(\text{new}) = \boldsymbol{w}_j^r(\text{old}) + \Delta \boldsymbol{w}_j^r(\text{new}) \qquad (4.25)$$

Dibandingkan dengan (4.4), kita melihat bahwa vektor koreksi $\Delta \boldsymbol{w}_j^r$ tidak hanya tergantung pada gradien tetapi juga pada nilai pada langkah iterasi sebelumnya. Konstan α disebut *faktor momentum* dan dalam praktek dipilih antara 0,1 dan 0,8. Untuk melihat pengaruh faktor momentum, mari kita lihat koreksi untuk sejumlah langkah iterasi yang berurutan. Pada tahap iterasi ke-1 kami mempunyai

$$\Delta \boldsymbol{w}_j^r(t) = \alpha \Delta \boldsymbol{w}_j^r(t-1) - \mu \boldsymbol{g}(t) \qquad (4.26)$$

dimana batas terakhir menunjukkan gradien. Untuk total T langkah iterasi yang berurutan kita mendapatkan

$$\Delta \boldsymbol{w}_j^r(T) = -\mu \sum_{t=0}^{T-1} \alpha^t \boldsymbol{g}(T-t) + \alpha^T \Delta \boldsymbol{w}_j^r(0) \qquad (4.27)$$

Sejak α<1, batas terakhir pergi mendekati nol setelah beberapa langkah iterasi dan *smoothing* (rata-rata) efek batas momentum menjadi jelas. Mari kita sekarang mengasumsikan bahwa algoritma tersebut berada pada titik yang rendah-lengkungan dari permukaan fungsi biaya dalam ruang berat. Kita kemudian dapat mengasumsikan bahwa gradien mendekati konstan selama beberapa langkah iterasi. Menerapkan ini, kita dapat menulis bahwa

$$\Delta w_j^r(T) \simeq -\mu(1+\alpha+\alpha^2+\alpha^3+\cdots)g = -\frac{\mu}{1-\alpha}g$$

Dengan kata lain, dalam kasus seperti pengaruh batas momentum untuk secara efektif meningkatkan pembelajaran konstan. Dalam praktek, peningkatan kecepatan dengan berkumpul faktor 2 atau bahkan lebih telah dilaporkan[Silv 90].

Sebuah variasi heuristik dari ini adalah dengan menggunakan nilai adaptif untuk faktor belajar μ, tergantung pada nilai-nilai fungsi biaya pada langkah iterasi yang berurutan. Sebuah prosedur mungkin adalah sebagai berikut: Biarkan *J(t)* menjadi nilai biaya pada langkah iterasi ke *t*. Jika *J(t) <J(t - 1)*, kemudian meningkatkan tingkat belajar dengan faktor r_i. Jika, di sisi lain, nilai baru biaya lebih besar dari yang lama oleh faktor *c*, kemudian menurun tingkat belajar dengan faktor r_d. Jika menggunakan nilai yang sama. Dalam ringkasan

$$\frac{J(t)}{J(t-1)} < 1, \quad \mu(t) = r_i\mu(t-1)$$

$$\frac{J(t)}{J(t-1)} > c, \quad \mu(t) = r_d\mu(t-1)$$

$$1 \le \frac{J(t)}{J(t-1)} \le c, \quad \mu(t) = \mu(t-1)$$

Nilai-nilai khas parameter yang diterapkan dalam praktek adalah r_i=1,05, r_d= 0.7. *c* = 1,04. Untuk langkah-langkah iterasi dimana kenaikan biaya, mungkin menguntungkan tidak hanya untuk menurunkan tingkat belajar tetapi juga untuk mengatur jangka momentum sama dengan 0. Lainnya menyarankan untuk tidak melakukan update dari pembobotan langkah ini.

Strategi lain untuk memperbarui faktor belajar μ diikuti dalam apa yang disebut aturan *delta-delta* dan dalam modifikasi ini aturan *delta-bar-delta* [Jaco 88]. Ide di sini adalah dengan menggunakan faktor belajar yang berbeda untuk setiap bobot dan meningkatkan faktor belajar tertentu jika gradien dari fungsi biaya sehubungan dengan yang sesuai berat memiliki tanda yang sama pada dua langkah iterasi berturut-turut. Sebaliknya, jika perubahan tanda ini merupakan indikasi mungkin adanya osilasi dan faktor belajar harus dikurangi. Sejumlah teknik alternatif untuk mempercepat konvergensi juga telah diusulkan. Dalam [Cich 93] ulasan yang lebih luas seperti teknik disediakan.

Pilihan lain untuk konvergensi yang lebih cepat adalah untuk membebaskan diri dari turunan gradien dan mengadopsi skema alternatif, biasanya atas biaya kompleksitas meningkat. Sejumlah teknik algoritmik tersebut telah muncul dalam literatur yang terkait. Sebagai contoh, [Kram 89, Barn 92, Joha 92] hadir skema algoritmik berdasarkan algoritma konjugasi gradien, [Batt 92, Rico 88, Barn 92, Watr 88] menyediakan skema dari keluarga Newton, [Palm 91, Sing 89] mengusulkan algoritma berdasarkan pendekatan Filter Kalman, dan [Bish95] skema berdasarkan algoritma Levenberg-Marquardt. Dalam banyak algoritma ini, elemen matriks Hessian butuh untuk dihitung, yaitu, yang kedua turunan dari fungsi biaya sehubungan dengan bobot

$$\frac{\partial^2 J}{\partial w_{jk}^q \partial w_{nm}^r}$$

Perhitungan matriks Hessian dilakukan dengan mengadopsi, sekali lagi, konsep propagasi balik (lihat juga Soal 4.12, 4.13). Lebih lanjut tentang isu-isu ini dapat ditemukan di [Hayk 99, Zura 92].

Skema populer yang didasarkan pada metode Newton adalah skema *quickprop* [Fahl90]. Ini adalah metode heuristik dan memperlakukan bobot seolah-olah mereka *quasi-independent*. Kemudian mendekati permukaan kesalahan, sebagai fungsi dari masing-masing bobot, oleh polinomial kuadratik. Jika hal ini minimum pada nilai yang masuk akal, yang terakhir digunakan sebagai bobot baru untuk iterasi, jika sejumlah heuristik digunakan. Bentuk biasa dari algoritma, untuk bobot di berbagai lapisan, adalah

$$\Delta w_{ij}(t) = \begin{cases} \alpha_{ij}(t)\Delta w_{ij}(t-1), & \text{if } \Delta w_{ij}(t-1) \neq 0 \\ \mu \frac{\partial J}{\partial w_{ij}}, & \text{if } \Delta w_{ij}(t-1) = 0 \end{cases} \qquad (4.28)$$

dimana

$$\alpha_{ij}(t) = \min \left\{ \frac{\frac{\partial J}{\partial w_{ij}}(t)}{\frac{\partial J}{\partial w_{ij}}(t-1) - \frac{\partial J}{\partial w_{ij}}(t)}, \alpha_{max} \right\} \qquad (4.29)$$

dengan nilai-nilai khas dari variabel yang terlibat menjadi $0{,}01 \leq \mu \leq 0.6$, $\alpha_{max} \approx 1{,}75$ [Cich 93]. Sebuah algoritma memiliki semangat yang sama untuk *quickprop* telah diusulkan di [Ried 93]. Hal ini melaporkan bahwa ini adalah secepat *quickprop* dan memerlukan sedikit penyesuaian parameter yang akan stabil.

4.8 PILIHAN FUNGSI BIAYA

Itu tidak akan mengejutkan bahwa fungsi biaya kuadrat terkecil di (4,6) bukan pilihan unik yang tersedia untuk pengguna. Tergantung pada masalah tertentu, fungsi biaya lainnya dapat menyebabkan hasil yang lebih baik. Mari kita lihat, misalnya, sekurang-kurangnya fungsi biaya kuadrat lebih hati-hati. Karena semua kesalahan dalam node-node output kuadrat pertama dan menyimpulkan, nilai kesalahan besar mempengaruhi proses belajar banyak lebih dari kesalahan kecil. Jadi, jika rentang dinamika output yang diinginkan tidak semua urutan yang sama, kriteria kuadrat terkecil akan menghasilkan bobot yang telah "belajar melalui proses penyediaan informasi yang tidak adil. Selanjutnya, dalam [Witt 88] terlihat bahwa untuk kelas masalah, algoritma *gradient descent* dengan kriteria kesalahan kuadrat dapat terjebak dalam minimum lokal dan gagal mencari solusi, meskipun (setidaknya) satu ada. Dalam konteks saat ini solusi adalah diasumsikan sebagai classifier yang mengklasifikasikan benar semua sampel pelatihan. Sebaliknya, terlihat bahwa ada kelas alternatif fungsi,

memenuhi kriteria tertentu, yang menjamin bahwa algoritma *gradient descent* menyatu untuk solusi tersebut, asalkan satu ada. Kelas ini fungsi biaya dikenal sebagai fungsi *well-formed*. Sekarang kita akan menyajikan fungsi biaya jenis ini, yang cocok untuk tugas-tugas pengenalan pola.

Jaringan multilayer melakukan pemetaan nonlinear dari vektor input x ke nilai output $\hat{y}_k = \phi_k(x; w)$ untuk masing-masing node output k = 1,2,. . . k_L, dimana ketergantungan pemetaan pada nilai-nilai bobot ditampilkan secara eksplisit. Dalam bab 3 kita telah melihat bahwa, jika kita mengadopsi fungsi kuadrat biaya termurah dan output yang diinginkan y_k adalah biner (milik atau tidak dalam kelas ω_k), kemudian untuk nilai optimal bobot ω^* output yang sesuai jaringan, $\hat{y}_k$, *adalah pengaruh optimal kuadrat terkecil dari probabilitas posterior* P (ω_k | x) (pertanyaan tentang baik atau buruk perkiraan ini akan menarik untuk kami). Pada titik ini kita akan mengadopsi interpretasi probabilistik dari output riil $\hat{y}_k$ sebagai dasar fungsi biaya kami akan dibangun. Mari kita berasumsi bahwa output yang diinginkan nilai-nilai, $\hat{y}_k$, bersifat independen variabel-variabel biner acak dan $\hat{y}_k$ yang masing-masing probabilitas posterior bahwa variabel acak adalah 1 [Hint90, Baum 88].

Fungsi biaya *cross-entropi* kemudian didefinisikan oleh

$$J = -\sum_{i=1}^{N}\sum_{k=1}^{k_L}(y_k(i)\ln\hat{y}_k(i) + (1 - y_k(i))\ln(1 - \hat{y}_k(i))) \qquad (4.30)$$

J mengambil nilai minimum ketika $y_k(i) = \hat{y}_k(i)$ dan untuk minimum nilai respon yang diinginkan biner adalah nol. Ada berbagai penafsiran dari fungsi biaya [Hint 90, Baum 88, Gish 90, Rich 91]. Mari kita mempertimbangkan, untuk contoh, vektor keluaran y(i) bila x(i) muncul di input. Ini terdiri dari 1 di node kelas benar dan nol untuk lainnya. Jika kita memperhitungkan bahwa probabilitas node k menjadi l(0) adalah $\hat{y}_k(i)(1 - \hat{y}_k(i))$ dan dengan mempertimbangkan node secara *independen*, kemudian

$$p(y) = \prod_{k=1}^{k_L}(\hat{y}_k)^{y_k}(1 - \hat{y}_k)^{1-y_k} \qquad (4.31)$$

dimana ketergantungan pada i telah tertekan untuk kenyamanan notasi. Kemudian mudah untuk memeriksa bahwa hasil J dari *loglikelihood* negatif pelatihan sepasang sampel. Jika $y_k(i)$ adalah probabilitas benar pada (0,1) kemudian mengurangkan nilai minimum dari J (4.30) menjadi

$$J = -\sum_{i=1}^{N}\sum_{k=1}^{k_L}\left(y_k(i)\ln\frac{\hat{y}_k(i)}{y_k(i)} + (1 - y_k(i))\ln\frac{1 - \hat{y}_k(i)}{1 - y_k(i)}\right) \qquad (4.32)$$

Untuk nilai biner y_k di atas masih berlaku jika kita menggunakan nilai batas 0 ln0 =0.

Hal ini tidak sulit untuk menunjukkan(Masalah 4.5) bahwa fungsi biaya *cross entropi tergantung pada kesalahan relatif dan bukan pada kesalahan mutlak, seperti pasangan kuadrat kecil, sehingga memberikan bobot yang sama untuk nilai kecil dan besar.* Selanjutnya, telah menunjukkan bahwa itu memenuhi kondisi fungsi terbentuk baik [Adal 97]. Akhirnya, dapat ditunjukkan bahwa mengadopsi fungsi biaya *cross entropi* dan nilai

biner untuk respon yang diinginkan, keluaran $\hat{y}_k$ sesuai dengan bobot optimal ω^* sesungguhnya perkiraan $P(\omega^* \mid x)$, seperti dalam kasus kuadrat terkecil [Hamp 90].

Keuntungan utama dari fungsi biaya *cross entropi* adalah bahwa hal itu menyimpang jika satu dari output menyatu ke kesalahan yang ekstrim, maka turunan gradien bereaksi cepat. Di sisi lain, fungsi biaya kuadrat kesalahan mendekati konstan dalam kasus ini, dan turunan gradien pada LS akan mengembara di dataran tinggi, meskipun kesalahan tidak mungkin kecil. Ini keuntungan dari fungsi biaya *cross entropi* adalah ditunjukkan dalam konteks pemerataan saluran dalam [Adal 97].

Sebuah hasil fungsi biaya yang berbeda jika kita memperlakukan $\hat{y}_k(i)$ dan y_k (i) benar dan sebagai probabilitas yang diinginkan, masing-masing. Kemudian ukuran kesamaan mereka diberikan oleh fungsi *cross-entropi* (Lampiran A)

$$J = -\sum_{i=1}^{N}\sum_{k=1}^{k_L} y_k(i)\ln\frac{\hat{y}_k(i)}{y_k(i)} \qquad (4.33)$$

Hal ini juga berlaku untuk nilai target biner (menggunakan bentuk batas). Namun, meskipun kita telah menafsirkan output sebagai probabilitas, tidak ada jaminan bahwa mereka jumlah untuk per satuan. Hal ini dapat dikenakan ke jaringan dengan mengadopsi fungsi alternatif aktivasi untuk node-node keluaran. Dalam [Brid 90] yang disebut *softmax* fungsi aktivasi yang disarankan, diberikan oleh

$$\hat{y}_k = \frac{\exp(v_k^L)}{\sum_{k'}\exp(v_{k'}^L)} \qquad (4.34)$$

Hal ini menjamin bahwa output terletak pada interval [0 1] dan bahwa jumlah mereka sampai dengan kesatuan (perhatikan bahwa berbeda dengan (4.32) probabilitas output tidak dianggap independen). Sangat mudah untuk menunjukkan, Soal 4.7, bahwa dalam kasus ini jumlah δ_j^L diperlukan oleh propagasi balik sama dengan $\hat{y}_k - y_k$.

Selain fungsi biaya *cross-entropi* dalam (4.33) sejumlah fungsi biaya alternatif telah diusulkan. Misalnya, dalam [Kara 92] generalisasi dari kesalahan fungsi biaya kuadrat digunakan dengan tujuan untuk mempercepat konvergensi. Tujuan lain adalah untuk meminimalkan kesalahan klasifikasi, yang setelah semua tujuan utama dalam pengenalan pola. Sejumlah teknik telah diusulkan dengan filsafat [Nede 93, Juan 92, Pado 95], yang dikenal sebagai *pembelajaran diskriminatif*. Keuntungan potensi dasar pembelajaran diskriminatif adalah bahwa pada dasarnya itu mencoba untuk memindahkan keputusan permukaan sehingga dapat mengurangi kesalahan klasifikasi. Untuk mencapai tujuan ini, menempatkan lebih menekankan pada kelas terbesar perkiraan probabilitas posteriori. Sebaliknya, fungsi biaya kesalahan kuadrat, misalnya, menetapkan sama penting bagi semua perkiraan probabilitas posterior. Dengan kata lain, ia mencoba untuk belajar lebih dari apa yang diperlukan untuk klasifikasi, yang dapat membatasi kinerjanya untuk jaringan ukuran tetap. Sebagian besar teknik pembelajaran diskriminatif menggunakan versi rapi dari kesalahan klasifikasi, sehingga dapat menerapkan diferensiasi berkaitan dengan pendekatan turunan gradien. Ini, tentu saja, bahaya bahwa prosedur minimisasi akan terjebak dalam

minimum lokal. Dalam [Mill 96] prosedur *deterministic annealing* digunakan untuk melatih jaringan, dengan potensi ditingkatkan untuk menghindari minimum lokal (lihat Bab 15).

Pilihan terakhir dari fungsi biaya tergantung pada masalah yang spesifik di bawah pertimbangan. Namun, seperti yang ditunjukkan dalam [Kaya 91], di sejumlah praktis situasi penggunaan alternatif, untuk kuadrat terkecil, fungsi biaya tidak selalu mengarah pada peningkatan kinerja substansial.

Sebuah Kerangka kerja Bayesian untuk Pelatihan Jaringan

Semua fungsi biaya sejauh ini dianggap bertujuan untuk komputasi satu set nilai optimal untuk parameter yang tidak diketahui dari jaringan. Sebuah pemikiran alternatif adalah melihat fungsi distribusi probabilitas dari bobot diketahui, ω, dalam bobot ruang. Gagasan di balik pendekatan ini berasal dari teknik inferensi Bayesian yang digunakan untuk perkiraan suatu pdf parametrik yang tidak diketahui, sebagaimana kita bahas dalam Bab 2. Mengikuti langkah-langkah dasar untuk jenis pelatihan jaringan, yang dikenal sebagai *pembelajaran Bayesian*, adalah (misalnya, [Mack 92a]):

Asumsikan model untuk distribusi prior p(ω) dari bobot. Ini harus agak luas dalam bentuk, untuk memberikan kesempatan yang sama untuk berbagai nilai agak besar.

Misalkan Y = {y (i), i = 1,2,..., N} menjadi set pelatihan vektor output yang diinginkan untuk data masukan yang diberikan set X = {x (i), i = 1,2,. . . N}. Asumsikan model untuk fungsi *likelihood* p(Y | ω), misalnya, Gaussian [4]. ([4]Strictly berbicara kita harus menulis P (Y | ω, X). Namun semua probabilitas dan pdf dikondisikan pada X dan kami menghilangkan untuk kenyamanan notasi).
Ini pada dasarnya model distribusi eror antara output yang benar dan nilai-nilai yang diinginkan, dan ini adalah tahap dimana data input pelatihan datang ke tempat kejadian.

Menggunakan teorema Bayes, kita memperoleh

$$p(\boldsymbol{w}|Y) = \frac{p(Y|\boldsymbol{w})p(\boldsymbol{w})}{p(Y)} \qquad (4.35)$$

dimana $p(Y) = \int p(Y|\boldsymbol{w})p(\boldsymbol{w})\,d\boldsymbol{w}$. Pdf posterior yang dihasilkan akan lebih berbentuk tajam sekitar nilai ω_0, karena telah belajar dari data pelatihan yang tersedia.

Menafsirkan output yang benar dari suatu jaringan, $\hat{y}_k = \phi_k(\boldsymbol{x}; \boldsymbol{w})$, karena masing-masing kelas probabilitas, dikondisikan pada input x dan vektor bobot ω, probabilitas kelas bersyarat dihitung dengan rata-rata lebih dari semua ω [Mack 92b]:

$$P(\omega_k|\boldsymbol{x}; Y) = \int \phi_k(\boldsymbol{x}; \boldsymbol{w})p(\boldsymbol{w}|Y)\,d\boldsymbol{w} \qquad (4.36)$$

Biaya komputasi utama yang terkait dengan jenis teknik ini dibutuhkan integrasi dalam ruang multidimensi. Ini bukan tugas yang mudah, dan berbagai implementasi praktis telah diusulkan dalam literatur. Lebih lanjut diskusi tentang masalah ini adalah di luar cakupan buku ini. Sebuah pengantar yang bagus untuk pembelajaran Bayesian, termasuk diskusi tentang implementasi praktis yang terkait, disediakan dalam [Bish 95].

4.9 PILIHAN UKURAN JARINGAN

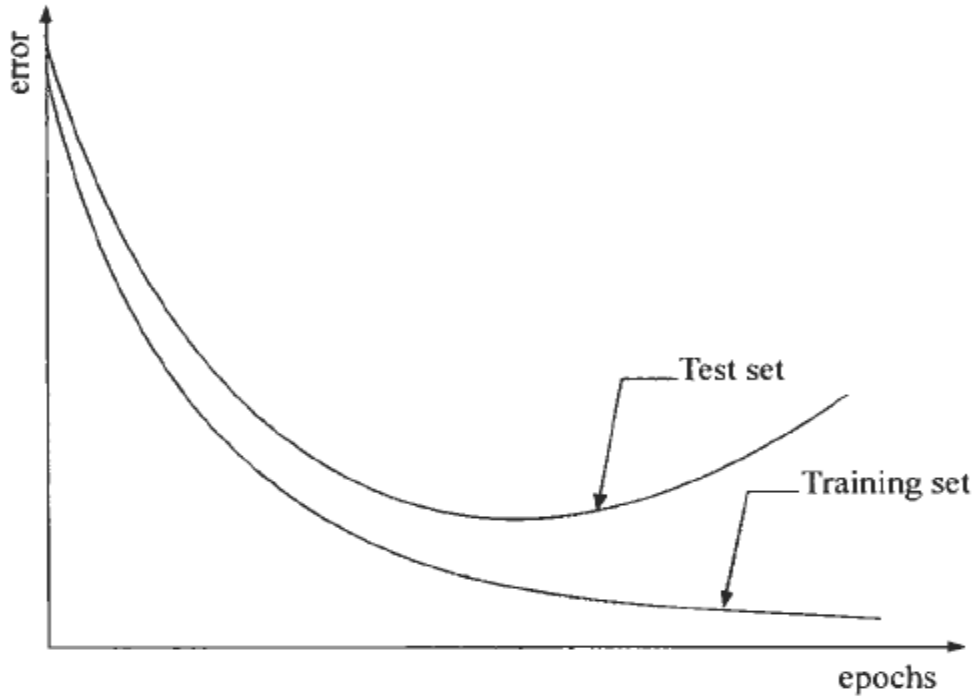
Pada bagian sebelumnya, kita asumsikan jumlah lapisan dan neuron untuk masing-masing layer untuk diketahui dan tetap. Bagaimana seseorang menentukan nomor yang sesuai lapisan dan neuron yang tidak menarik bagi kita. Tugas ini akan menjadi fokus utama kami sekarang.

Satu jawaban untuk masalah itu bisa untuk memilih ukuran jaringan cukup besar dan meninggalkan pelatihan untuk memutuskan tentang bobot. Sebuah pemikiran sederhana mengungkapkan bahwa pendekatan semacam ini agak naif. Selain kompleksitas komputasi terkait masalah, ada alasan utama mengapa ukuran jaringan harus dijaga sekecil mungkin. Hal ini ditentukan oleh kemampuan generalisasi yang jaringan harus memiliki. Sebagaimana telah ditunjukkan dalam Bagian 3.6, istilah generalisasi mengacu pada kemampuan jaringan syaraf multilayer (dan setiap klasifikasi) untuk mengklasifikasikan vektor fitur benar yang tidak disajikan selama tahap pelatihan-yaitu, kemampuan jaringan untuk memutuskan pada data yang tidak diketahui, berdasarkan apa yang telah dibelajari dari set pelatihan. Mengambil untuk diberikan terbatas (dan dalam banyak kasus kecil) N jumlah pasangan pelatihan, jumlah parameter bebas (bobot synaptic) diperkirakan harus *(a) yang cukup besar untuk mempelajari apa membuat "mirip" vektor fitur di dalam kelas masing-masing dan pada saat yang sama apa yang membuat satu kelas berbeda dari lainnya dan (b) cukup kecil, sehubungan dengan N, sehingga tidak dapat mempelajari perbedaan mendasar antara data kelas sama.* Ketika jumlah parameter bebas adalah besar, jaringan cenderung untuk beradaptasi dengan rincian tertentu dari set data pelatihan khusus yang ditetapkan. Hal ini dikenal sebagai *overfitting* dan menyebabkan kinerja generalisasi yang buruk, ketika jaringan dipanggil untuk beroperasi pada fitur vektor yang tidak diketahui. Sebagai kesimpulan, jaringan harus memiliki terkecil

ukuran yang mungkin untuk menyesuaikan bobot kepada keteraturan terbesar di data dan mengabaikan yang lebih kecil, yang mungkin juga hasil dari pengukuran *noise*. Beberapa teori menyentuh tentang aspek generalisasi dari suatu klasifikasi akan disajikan dalam Bab 5, ketika kita membahas dimensi *Vapnik-Chernovenkis*. Dilema variasi bias, dibahas dalam Bab 3, adalah sisi lain dari masalah yang sama.

Adaptasi parameter bebas pada karakter pelatihan khusus yang juga dapat terjadi sebagai akibat dari *overtraining* (misalnya, lihat [Chau 90]). Mari kita asumsikan bahwa kita mampu membayar kemewahan memiliki satu set data pelatihan yang besar. Kami membagi satu set ini menjadi dua himpunan bagian, satu untuk pelatihan dan satu untuk tes. Yang terakhir ini dikenal sebagai *validasi* atau *uji*. Gambar 4.14 menunjukkan kecenderungan dua kurva dari kesalahan output sebagai fungsi dari langkah-langkah iterasi. Salah satu sesuai dengan set pelatihan, dan kita amati bahwa kesalahan terus menurun sebagai konvergensi bobot. Yang lain sesuai dengan kesalahan set validasi. Awalnya kesalahan berkurang, tetapi pada beberapa saat kemudian tahap itu mulai meningkat. Hal ini karena bobot, dihitung dari set pelatihan, beradaptasi ke keistimewaaan set pelatihan khusus, sehingga mempengaruhi generalisasi kinerja jaringan. Perilaku ini dapat digunakan dalam praktek untuk menentukan titik di mana proses belajar iterasi harus berhenti. Ini adalah titik dimana dua kurva mulai berangkat. Namun, metodologi ini berasumsi keberadaan sejumlah besar set data, yang tidak biasanya terjadi dalam praktek.

Selain generalisasi, faktor kinerja lain juga permintaan untuk menjaga ukuran dari jaringan sekecil mungkin. Jaringan kecil merupakan komputasi lebih cepat dan lebih murah untuk membangun. Selain itu, kinerja mereka lebih mudah untuk memahami, yang penting dalam beberapa aplikasi kritis.



Gambar 4.14**:** Kecenderungan dari kesalahan output versus ilustrasi jumlah *epochs overtraining* dari set pelatihan

Pada bagian ini kita fokus pada metode yang pilih nomor yang sesuai dari parameter bebas, dengan kriteria tertentu dan untuk dimensi tertentu dari ruang vektor input. Yang terakhir adalah sangat penting, karena dimensi data input terkait keraguan dengan sejumlah parameter bebas untuk digunakan, sehingga juga mempengaruhi generalisasi properti. Kami akan datang ke isu-isu terkait dengan pengurangan input ruang dimensi dalam Bab 5.

Pendekatan yang paling banyak digunakan untuk memilih ukuran jaringan multilayer datang di bawah salah satu kategori berikut:

- *Metode analitis.* Kategori ini menggunakan teknik aljabar atau statistik untuk menentukan jumlah parameter bebas.
- *Teknik pemangkasan.* Sebuah jaringan besar awalnya dipilih untuk pelatihan dan kemudian jumlah parameter bebas adalah berturut-turut berkurang, menurut aturan yang dipilih sebelumnya.
- *Teknik konstruktif.* Sebuah jaringan kecil awalnya dipilih dan neuron-neuron adalah berturut-turut menambahkan, berdasarkan aturan belajar yang tepat diadopsi.

Estimasi Aljabar dari Jumlah Parameter Bebas

Kita telah bahas dalam Bagian 4.3.1 kemampuan dari multilayer perceptron, dengan satu lapisan tersembunyi dan satuan dari tipe McCulloch-Pitts, untuk membagi masukan ruang dimensi *l* ke beberapa daerah *polyhedral*. Ini adalah hasil persimpangan dari *hyperplanes* yang dibentuk oleh neuron. Dalam [Mirc 89] itu menunjukkan bahwa dalam ruang dimensi *l* multilayer perceptron dengan sebuah lapisan tersembunyi tunggal dengan neuron K dapat membentuk *maksimum* daerah M polyhedral dengan M diberikan oleh

$$M = \sum_{m=0}^{l} \binom{K}{m}, \text{ where } \binom{K}{m} = 0, \quad \text{for } K < m \tag{4.37}$$

dan

$$\binom{K}{m} \equiv \frac{K!}{m!(K-m)!}$$

Misalnya, jika l = 2, dan K = 2, hasil ini M = 4, dengan demikian masalah XOR, dengan M = 3 (<4), dapat diselesaikan dengan dua neuron. Kerugian metode ini adalah bahwa hal itu statis dan tidak mempertimbangkan fungsi biaya yang digunakan serta prosedur pelatihan.

Teknik Pemangkasan

Teknik-teknik ini memulai pelatihan jaringan cukup besar dan kemudian mereka hapus, dalam prosedur bertahap, parameter bebas sedikit berpengaruh pada fungsi biaya. Ada dua arah metodologi utama:

Metode berdasarkan perhitungan parameter sensitivitas. Mari kita ambil sebagai contoh teknik yang disarankan dalam [Lecu 90]. Dengan menggunakan ekspansi deret Taylor, variasi dikenakan pada fungsi biaya dengan gangguan parameter adalah

$$\delta J = \sum_i g_i \delta w_i + \frac{1}{2} \sum_i h_{ii} \delta w_i^2 + \frac{1}{2} \sum_{\substack{i,j \\ i \neq j}} h_{ij} \delta w_i \delta w_j$$

$$+ \text{higher order terms}$$

dimana

$$g_i = \frac{\partial J}{\partial w_i}, \qquad h_{ij} = \frac{\partial^2 J}{\partial w_i \partial w_j}$$

dan i, j berjalan atas semua bobot. Turunan dapat dihitung melalui metodologi *backpropagation* (Soal 4.13). Dalam prakteknya, perhitungan turunan dilakukan setelah beberapa periode awal pelatihan. Hal ini memungkinkan kita untuk mengadopsi asumsi bahwa titik dekat minimum telah tercapai dan turunan pertama dapat diatur sama dengan nol. Sebuah penyederhanaan komputasi lebih lanjut adalah mengasumsikan bahwa matriks Hessian diagonal. Berdasarkan asumsi-asumsi, biaya fungsi sensitivitas sekitar diberikan oleh

$$\delta J = \frac{1}{2} \sum_i h_{ii} \delta w_i^2 \tag{4.38}$$

dan kontribusi setiap parameter ditentukan oleh nilai *saliency* s_i, diberikan kira-kira oleh

$$s_i = \frac{h_{ii} w_i^2}{2} \qquad (4.39)$$

di mana kita berasumsi bahwa bobot nilai ω_i diubah menjadi nol. Pemangkasan sekarang dicapai dengan cara yang berulang sesuai dengan langkah-langkah berikut:

- Jaringan ini dilatih dengan menggunakan algoritma *backpropagation* untuk sejumlah langkah-langkah iterasi sehingga fungsi biaya berkurang untuk persentase yang cukup.
- Untuk perkiraan bobot saat ini, nilai *saliency* masing-masing dihitung dan bobot dengan *saliency* kecil dihapus.
- Proses pelatihan dilanjutkan dengan bobot yang tersisa dan proses diulangi setelah beberapa langkah iterasi. Proses ini berhenti ketika yang dipilih menghentikan kriteria terpenuhi.

Dalam [Hass 93] penuh matriks Hessian sudah bekerja selama prosedur pemangkasan. Harus ditekankan bahwa walaupun konsep *backpropagation* hadir dalam teknik, prosedur pembelajaran yang jelas berbeda dari algoritma pelatihan *backpropagation* Bagian 4.6. *Di sana, sejumlah parameter bebas telah ditetapkan seluruh pelatihan. Sebaliknya, filsafat di sini adalah justru sebaliknya.*

Metode berdasarkan regularisasi fungsi biaya. Metode ini mencapai pengurangan yang dari ukuran besar awalnya jaringan dengan memasukkan *penalty term* di fungsi biaya. Fungsi biaya sekarang memiliki bentuk

$$J = \sum_{i=1}^{N} \mathcal{E}(i) + \alpha \mathcal{E}_p(\boldsymbol{w}) \qquad (4.40)$$

Istilah pertama adalah kinerja fungsi biaya, dan dipilih sesuai dengan apa yang telah kita bahas (misalnya, kuadrat terkecil, *cross entropy*). Yang kedua tergantung pada vektor bobot, dan ini dipilih untuk *mendukung nilai-nilai kecil untuk bobot.* Konstan α adalah apa yang disebut *parameter regularisasi*, dan kontrol relatif makna dari kedua istilah. Sebuah bentuk populer untuk *penalty term* adalah

$$\mathcal{E}_p(\boldsymbol{w}) = \sum_{k=1}^{K} h(w_k^2) \qquad (4.41)$$

dengan K sebagai jumlah bobot dalam jaringan dan h(.) suatu fungsi terdiferensialkan tepat yang dipilih. Menurut pilihan, bobot yang tidak memberikan kontribusi signifikan dalam pembentukan output jaringan tidak mempengaruhi banyak istilah pertama dari fungsi biaya. Oleh karena itu, keberadaan *penalty term* mendorong mereka untuk nilai kecil. Jadi, pemangkasan dicapai. Dalam prakteknya, ambang dipilih sebelumnya dan bobot dibandingkan setelah sejumlah langkah iterasi. Bobot yang menjadi lebih kecil dikeluarkan, dan proses dilanjutkan. Tipe pemangkasan ini dikenal sebagai *eliminasi bobot*. Fungsi h(.) dapat mengambil berbagai bentuk. Misalnya, dalam [Wein 90] berikut ini disarankan:

$$h(w_k^2) = \frac{w_k^2}{w_0^2 + w_k^2} \qquad (4.42)$$

dimana ω_0 adalah parameter yang telah dipilih sebelumnya untuk persatuan. Pengamatan lebih dekat dari *penalty term* ini mengungkapkan bahwa ia pergi ke nol sangat cepat untuk nilai ω_0; sehingga bobot tersebut menjadi tidak signifikan. Sebaliknya, *penalty term* cenderung kesatuan untuk $\omega_k > \omega_0$.

Sebuah variasi (4.41) adalah termasuk dalam fungsi biaya *penalty term* lain yang menyukai nilai kecil y_k^r, yaitu, output neuron kecil. Seperti teknik mengakibatkan penghapusan neuron tidak signifikan serta bobot. Sebuah ringkasan dan diskusi tentang berbagai teknik pemangkasan dapat ditemukan di [Refe 91, Russ 93].

Teknik Konstruktif

Di Bagian 4.5 kita telah membahas teknik-teknik tersebut untuk pelatihan jaringan syaraf. Namun, fungsi aktivasi adalah fungsi unit step dan juga penekanan diletakkan pada mengklasifikasikan input data semua pelatihan dengan benar dan bukan pada generalisasi sifat dari jaringan yang dihasilkan. Dalam [Fahl 90] alternatif teknik konstruktif untuk pelatihan jaringan syaraf tiruan, dengan lapisan tunggal tersembunyi fungsi aktivasi sigmoid, diusulkan, yang dikenal sebagai *korelasi kaskade*. Jaringan dimulai dengan unit input dan output saja. Neuron tersembunyi ditambahkan satu per satu dan yang terhubung ke jaringan dengan dua jenis bobot. Tipe pertama menghubungkan unit baru dengan node input maupun output dari sebelumnya ditambahkan neuron tersembunyi. Setiap kali sebuah neuron tersembunyi baru akan ditambahkan dalam jaringan, bobot ini dilatih sehingga memaksimalkan hubungan antara unit baru output dan sinyal kesalahan residu dalam jaringan output sebelum penambahan unit baru. Setelah neuron ditambahkan, bobot ini dihitung sekali dan kemudian mereka tetap. Jenis kedua bobot sinaptik terhubung yang baru ditambah neuron dengan node output. Bobot ini yang tidak tetap dan dilatih adaptif, setiap kali neuron baru dipasang, untuk meminimalkan jumlah kuadrat kesalahan fungsi biaya. Prosedur akan berhenti bila kinerja jaringan memenuhi tujuan yang ditentukan. Sebuah diskusi tentang teknik konstruktif dengan penekanan pada pengenalan pola dapat ditemukan di [Pare 00].

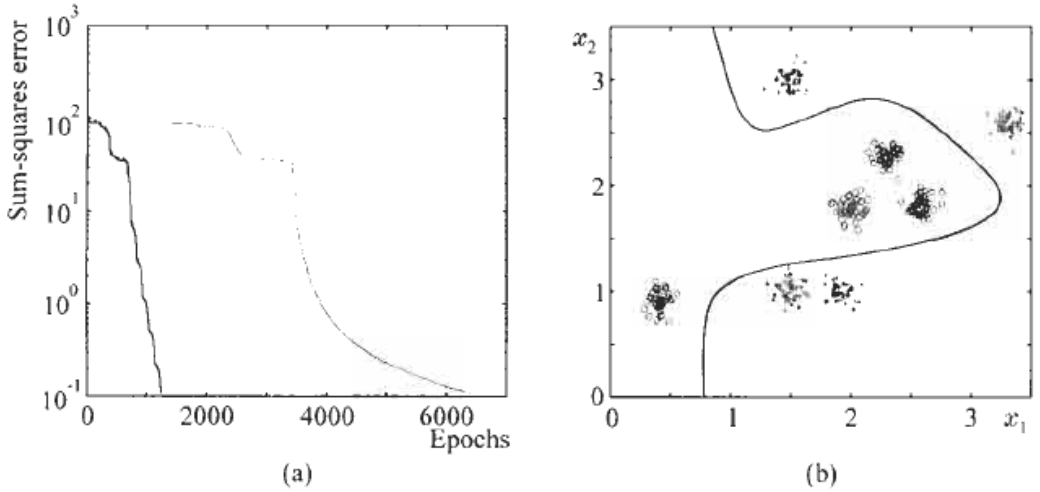
4.10 CONTOH SIMULASI

Pada bagian ini, kemampuan suatu multilayer perceptron untuk mengklasifikasikan kelas nonlinear terpisah. Tugas klasifikasi terdiri dari dua kelas yang berbeda, masing-masing himpunan dari empat daerah dalam ruang dua dimensi. Setiap wilayah terdiri dari vektor acak terdistribusi normal dengan komponen statistik independen dan masing-masing dengan varians $\alpha^2 = 0.08$. Nilai rata-rata berbeda untuk masing-masing daerah. Khususnya, daerah kelas dinotasikan dengan "0" (lihat Gambar 4.15) terbentuk sekitar rata-rata vektor

$$[0.4, 0.9]^T, [2.0, 1.8]^T, [2.3, 2.3]^T, [2.6, 1.8]^T$$

dan hal itu dari kelas dinotasikan dengan "+" di sekitar nilai

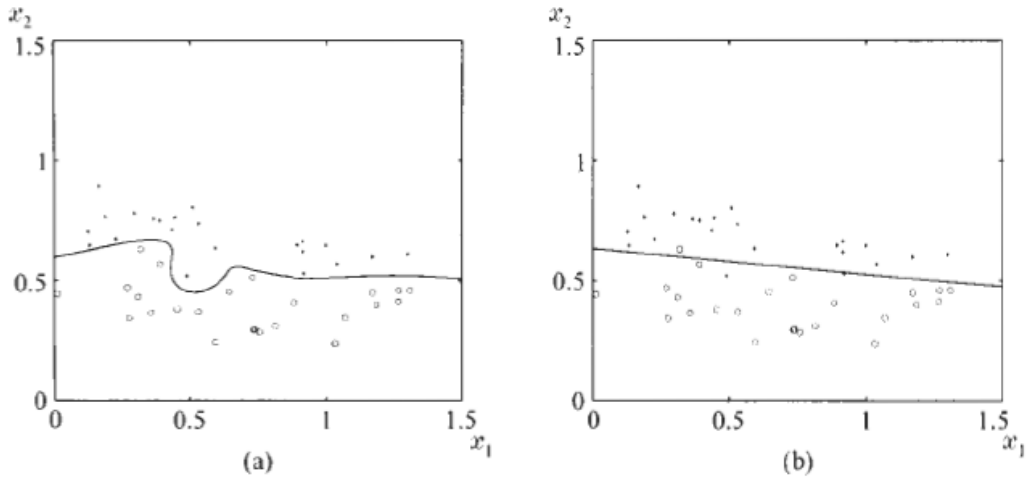
$[1.5, 1.0]^T, [1.9, 1.0]^T, [1.5, 3.0]^T, [3.3, 2.6]^T$



Gambar 4.15**:** (a) Kesalahan kurva konvergensi untuk momentum adaptif dan algoritma momentum. Perhatikan bahwa momentum adaptif mengarah lebih cepat konvergensi. (B) Kurva keputusan dibentuk oleh multilayer perceptron.

Sebanyak 400 vektor pelatihan dihasilkan, 50 dari setiap distribusi. Sebuah multilayer perseptron dengan tiga neuron di pertama dan dua neuron di kedua lapisan tersembunyi yang digunakan, dengan neuron output tunggal. Fungsi aktivasi satu logistik dengan a = 1 dan output yang diinginkan 1 dan 0, masing-masing, untuk dua kelas. Dua algoritma yang berbeda digunakan untuk pelatihan, yaitu momentum dan momentum adaptif. Setelah percobaan beberapa yang algoritmik parameter yang digunakan adalah (a) untuk momentum μ= 0,05, α= 0,85, dan (b) untuk momentum adaptif μ= 0,01, α= 0,85 r_i= 1.05, c=1.05, r_d=0.7. Bobot tersebut diinisialisasi dengan sebuah distribusi pseudorandom seragam antara 0 dan 1. Gambar 4.15a menunjukkan kurva konvergensi error output masing-masing untuk dua algoritma sebagai fungsi dari jumlah *epochs* (masing-masing *epoch* terdiri dari 400 vektor fitur pelatihan). Kurva masing-masing dapat dianggap khas dan algoritma momentum adaptif mengarah pada konvergensi lebih cepat. Kedua kurva sesuai dengan operasi mode batch. Gambar 4.15b menunjukkan keputusan yang dihasilkan permukaan dengan menggunakan bobot diperkirakan dari pelatihan momentum adaptif. Setelah bobot jaringan telah diperkirakan, permukaan keputusan dapat mudah ditarik. Untuk tujuan ini, kotak dua dimensi dibangun atas wilayah tersebut, dan titik-titik grid diberikan sebagai input ke jaringan, baris demi baris. Permukaan keputusan dibentuk oleh titik-titik di mana output dari jaringan berubah dari 0 ke 1 atau sebaliknya.

Percobaan kedua dilakukan untuk menunjukkan pengaruh pemangkasan. Gambar 4.16 menunjukkan permukaan yang dihasilkan keputusan memisahkan sampel dari dua kelas, dinotasikan dengan "+" dan "0" masing-masing. Gambar 4.16a sesuai multilayer perceptron (MLP) dengan dua lapisan tersembunyi dan 20 neuron di masing-masing dari mereka, sebesar total 480 bobot. Pelatihan dilakukan melalui



Gambar 4.16: Kurvasanu keputusan (a) sebelum pemangkasan (b) setelah pemangkasan

algoritma *backpropagation.* Sifat *overfitting* dari (a) kurva yang dihasilkan mudah diamati. Gambar 4.16b sesuai dengan MLP yang sama dilatih dengan algoritma pemangkasan. Secara khusus, dengan metode berdasarkan sensitivitas parameter yang digunakan, pengujian nilai *saliency* dari bobot setiap 100 *epoch* dan bobot menghapus dengan nilai *saliency* di bawah ambang batas yang dipilih. Akhirnya, hanya 25 dari 480 bobot adalah kiri, dan kurva ini disederhanakan ke garis lurus.

4.11 JARINGAN DENGAN BERBAGI BOBOT

Satu masalah utama yang dihadapi dalam banyak aplikasi pengenalan pola adalah transformasi invariance. Ini berarti sistem pengenalan pola harus benar mengklasifikasikan, dari transformasi dilakukan independen pada ruang input, seperti translasi, rotasi, dan skala. Sebagai contoh, karakter "5" seharusnya "terlihat sama" untuk sistem OCR, terlepas dari, orientasi posisinya, dan ukuran. Ada sejumlah cara untuk pendekatan masalah ini. Salah satunya adalah untuk memilih sesuai fitur vektor, yang berubah dalam transformasi tersebut. Ini akan menjadi salah satu tujuan utama kami dalam Bab 7. Cara lain adalah membuat klasifikasi bertanggung jawab untuk itu dalam bentuk *built-in constraints*. *Berbagi bobot adalah seperti kendala, yang memaksa koneksi tertentu dalam jaringan untuk memiliki bobot sama* .

Salah satu tipe jaringan di mana konsep berbagi bobot telah diadopsi disebut *jaringan orde tinggi*. Ini adalah perceptrons multilayer dengan aktivasi fungsi yang bekerja pada nonlinear, bukan linear, kombinasi parameter input. Keluaran dari neuron sekarang dalam bentuk

$$f(v) = f\left(w_0 + \sum_{i} w_i x_i + \sum_{j,k} w_{jk} x_j x_k\right)$$

Hal ini dapat digeneralisasi untuk mencakup produk orde tinggi. Mari kita asumsikan bahwa input ke jaringan berasal dari dua dimensi grid (gambar). Setiap titik dari grid yang berkaitan dengan suatu x_i tertentu dan masing-masing pasangan (x_i, x_j) ke ruas garis. Invarian untuk tanslasi dibangun oleh bobot $\omega_{jk} = \omega_{rs}$, jika segmen garis masing-masing, yang didefinisikan oleh titik (x_j, x_k) dan (x_r, x_s), adalah dari gradien yang sama. Invarian untuk rotasi dapat dibangun oleh bobot berbagi sesuai dengan segmen dengan panjang sama. Tentu saja, semua ini mengacu pada ketidakakuratan yang disebabkan oleh kekasaran resolusi dari grid. Agar jaringan orde tinggi dapat mengakomodasi transformasi yang lebih kompleks [Kana 92, Pera

92, Delo 94]. Karena berbagi bobot, jumlah parameter bebas untuk optimasi secara substansial berkurang. Namun, harus menunjukkan bahwa, sejauh ini, jaringan tersebut belum banyak digunakan dalam praktek. Suatu tipe khusus dari jaringan yang disebut *model propagasi balik lingkaran.*

Hasilnya

$$f(v) = f\left(w_0 + \sum_i w_i x_i + w_s \sum_i x_i^2\right)$$

Peningkatan jumlah parameter sekarang kecil, dan dalam [Ride 97] diklaim bahwa keterlibatan istilah nonlinier menawarkan jaringan meningkat representasi daya tanpa mempengaruhi kemampuan generalisasi.

Selain jaringan orde yang lebih tinggi, berbagi bobot telah digunakan untuk invarian dikenakan pada jaringan orde pertama digunakan untuk aplikasi khusus [Fuku 82, Rume 86, Fuku 92, Lecu 89]. Yang terakhir, misalnya, suatu sistem tulisan tangan pengenalan kode zip. Ini adalah struktur hirarki dengan tiga lapisan tersembunyi dan input tingkat abu-abu dari pixel gambar. Node-node dalam dua lapisan pertama berupa kelompok array dua dimensi yang dikenal sebagai *peta fitur*. Setiap node dalam diberikan peta input dari area jendela khusus dari lapisan sebelumnya, yang dikenal sebagai *bidang reseptif.* Invarian dikenakan dengan terkait node dalam peta yang sama, melihat bidang reseptif yang berbeda, untuk berbagi bobot. Jadi, jika sebuah obyek bergerak dari satu input bidang reseptif terhadap yang lain, jaringan menanggapi dengan cara yang sama.

4.12 GENERALISASI KLASIFIKASI LINIER

Pada Bagian 4.3, mengenai permasalahan XOR nonlinear terpisah, kita melihat bahwa neuron lapisan tersembunyi melakukan pemetaan yang mengubah masalah ke linier terpisah. Pemetaan yang sebenarnya

$$x \rightarrow y$$

dengan

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} f(g_1(x)) \\ f(g_2(x)) \end{bmatrix} \qquad (4.43)$$

dimana f (.) adalah fungsi aktivasi dan $g_i(.)$ (x), i = 1,2, kombinasi linier dari input yang dilakukan oleh masing-masing neuron. Ini akan menjadi titik *kickoff* kita untuk bagian ini.

Mari kita mempertimbangkan vektor fitur kami berada di *l* ruang dimensi $R^{'}$ dan menganggap bahwa mereka termasuk salah satu dari dua kelas A, B, yang terpisah nonlinear. Biarkan $f_l(.)$, $f_2(.)$. . . , $f_k(.)$ akan nonlinear (dalam kasus umum) fungsi

$$f_i : \mathcal{R}^l \rightarrow \mathcal{R}, \quad i = 1, 2, \ldots, k$$

yang menentukan pemetaan $x \in R' \rightarrow y \in R^k$

$$y \equiv \begin{bmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_k(x) \end{bmatrix}$$

Tujuan kami sekarang adalah untuk menyelidiki apakah ada nilai yang sesuai untuk k dan fungsi f_i sehingga kelas A, B linear terpisah di ruang k dimensi vektor y. Dengan kata lain, kami menyelidiki apakah terdapat ruang k dimensi di mana kita dapat membangun *hyperplane* $\omega \epsilon R^k$ sehingga

$$w_0 + \boldsymbol{w}^T \boldsymbol{y} > 0, \quad \boldsymbol{x} \in A \tag{4.45}$$

$$w_0 + \boldsymbol{w}^T \boldsymbol{y} < 0, \quad \boldsymbol{x} \in B \tag{4.46}$$

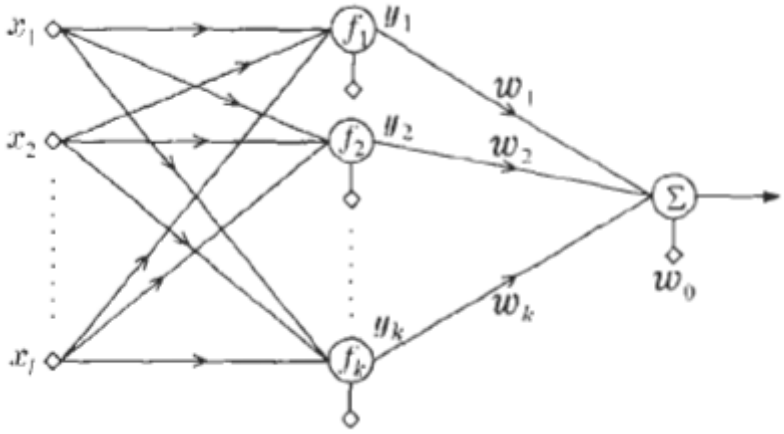
Dengan asumsi bahwa di ruang asli kelas dua yang dipisahkan oleh (nonlinear) *hypersurface* g(x) = 0, hubungan (4.45), (4.46) pada dasarnya setara dengan yang kurang lebih sama nonlinear g (x) sebagai kombinasi linear dari $f_i(x)$, yaitu,

$$g(\boldsymbol{x}) = w_0 + \sum_{i=1}^{k} w_i f_i(\boldsymbol{x}) \tag{4.47}$$

Ini adalah masalah khas pendekatan fungsi yang dipilih sebelumnya kelas *fungsi interposi* $f_i(.)$. Ini adalah tugas baik belajar pada analisis numerik, dan sejumlah fungsi interpolasi yang berbeda telah diusulkan (eksponensial, polinomial, Tchebyshev, dll). Dalam bagian berikutnya kita akan fokus pada kelas dua fungsi, yang telah banyak digunakan dalam pengenalan pola.

Setelah f_i fungsi telah dipilih, masalahnya menjadi desain khas dari klasifikasi linear, yaitu, untuk memperkirakan bobot ω_i dalam ruang k-dimensi. Ini membenarkan *klasifikasi linier generalisasi*. Gambar 4.17 menunjukkan sesuai blok diagram. Lapisan pertama dari perhitungan melakukan pemetaan untuk ruang y; lapisan kedua melakukan perhitungan keputusan *hyperplane*. Dengan kata lain, (4.47) sesuai dengan *jaringan dua-lapisan* dimana node-node dari lapisan tersembunyi memiliki fungsi aktivasi yang berbeda, $f_i(.)$ i = 1,2,...,K. Untuk masalah kelas-M kita perlu merancang M vektor ω_r bobot tersebut, r = 1,2,. . . , M, satu untuk setiap kelas, dan pilih kelas r menurut output maksimum $\boldsymbol{w}_r^T \boldsymbol{y} + w_{r0}$.

Pertama kita akan mencoba untuk membenarkan dugaan kita, bahwa dengan pergi ke ruang dimensi yang lebih tinggi tugas klasifikasi dapat berubah menjadi satu linier dan kemudian mempelajari alternatif populer untuk pilihan fungsi f_i (.).



Gambar 4.17**:** Klasifikasi Generalisasi Linear

4.13 KAPASITAS DARI RUANG *l*-DIMENSI DALAM DIKOTOMI LINEAR

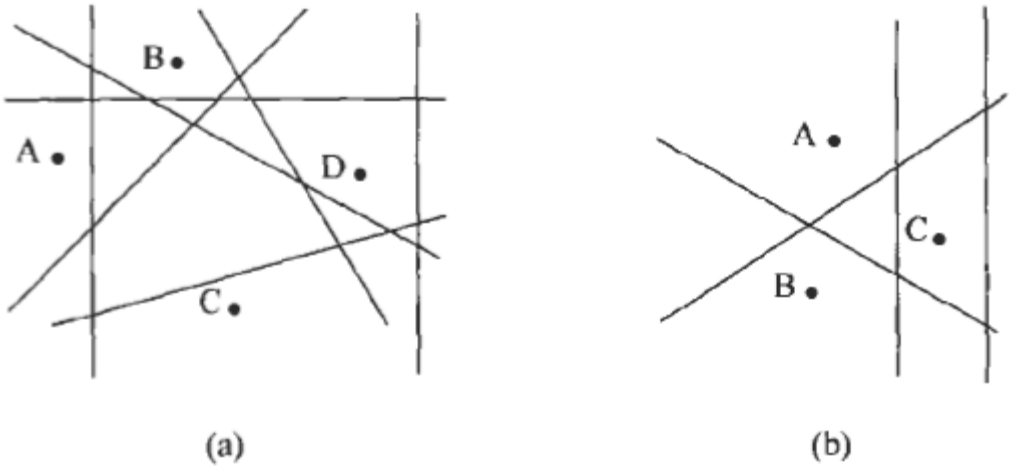
Mari kita perhatikan titik N dalam ruang *l*-dimensi. Kami akan mengatakan bahwa titik ini berada *di posisi umum atau didistribusikan dengan baik* jika tidak ada subset dari *l*+1 dari mereka yang terletak pada sebuah *hyperplane* (*l*-1)-dimensi. Seperti dalam ruang 2-dimensi yang memiliki tiga titik di garis lurus (hyperplane 1-dimensi). Jumlah O(N, 1) dari kelompok yang dapat dibentuk oleh (*l*- \1) *hyperplanes* dimensi untuk memisahkan titik N dalam dua kelas, mengambil semua kombinasi yang mungkin, diberikan oleh ([Cove 65] dan Soal 4.18):

$$O(N, l) = 2\sum_{i=0}^{l}\binom{N-1}{i} \qquad (4.48)$$

dimana

$$\binom{N-1}{i} = \frac{(N-1)!}{(N-1-i)!i!} \qquad (4.49)$$

Masing-masing dua pengelompokan kelas juga dikenal sebagai *dikotomi* (linear). Dari sifat koefisien binomial, ternyata bahwa untuk N≤ *l* + 1, O(N, *l*) = 2^N. Gambar 4.18 menunjukkan dua contoh *hyperplanes* seperti yang dihasilkan pada O(4,2) = 14 dan O(3,2) = 8 pengelompokan dua kelas, masing-masing. Tujuh baris Gambar 4.18a membentuk kelompok berikut. [(ABCD)], [A, (BCD)], [B, (ACD)], [C, (ABD)], [D, (ABC)], [(AB), (CD)], [(AC), (BD)]. Setiap pengelompokan sesuai dengan dua kemungkinan. Sebagai contoh, (ABCD) dapat termasuk kelas lain ω_1 atau ω_2. Demikian, jumlah kombinasi penempatan empat titik pada

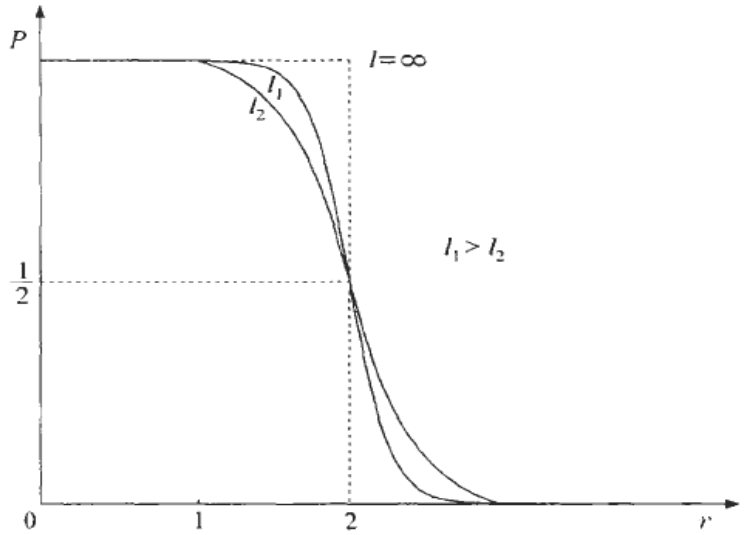


Gambar 4.18**:** Jumlah dikotomi linear (a) untuk empat dan (b) untuk tiga titik.

ruang-2 dimensi dalam dua *kelas terpisah* secara linear adalah 14. Jumlah ini jelas lebih kecil dari jumlah total kombinasi dari penempatan titik N dalam dua kelas, yang dikenal sebagai 2^N. Hal ini karena juga melibatkan kombinasi nonlinear terpisah. Dalam kasus contoh kita, ini adalah 16, yang muncul dari dua kemungkinan tambahan dari pengelompokan [(AD), (SM)]. Kita sekarang siap untuk menulis probabilitas (prosentase) dari pengelompokan titik N di ruang *l*-dimensi di dua kelas linear terpisah [Cove 65]. Ini diberikan oleh:

$$P_N^l = \frac{O(N,l)}{2^N} = \begin{cases} \frac{1}{2^{N-1}} \sum_{i=0}^{l} \binom{N-1}{i} & N > l+1 \\ 1 & N \le l+1 \end{cases} \qquad (4.50)$$

Sebuah cara praktis untuk mempelajari ketergantungan P_N^l pada N dan 1 adalah mengasumsikan bahwa N = r(*l*+1) dan menyelidiki kemungkinan untuk berbagai nilai r. Kurva di Gambar 4.19 menunjukkan kemungkinan memiliki kelas dipisahkan secara linear untuk berbagai nilai *l*. Hal ini mudah diamati bahwa ada dua daerah, satu ke kiri r = 2, yakni, N= 2 (*l* +1) dan satu ke kanan. Selanjutnya, semua kurva melalui titik (P_N^l, r) = (1/2, 2), karena fakta bahwa 0 (2*l*+2, l) = 2^{2l+1} (Soal 4.19). Transisi dari satu daerah dengan daerah lain menjadi lebih tajam sebagai $l \to \infty$. Jadi, untuk besar nilai 1 dan jika N<2 (*l*+1) probabilitas dari dua kelompok titik-titik N menjadi kesatuan pendekatan terpisah secara linier. Sebaliknya benar jika N>2 (*l*+1). Dalam prakteknya, kita tidak mampu membayar kemewahan nilai-nilai yang sangat besar dari N dan *l*, temuan kami menjamin bahwa jika kita diberikan titik N, maka pemetaan ke dalam ruang dimensi lebih tinggi meningkatkan kemungkinan menemukan mereka dalam kelas dua kelompok linear terpisah.



Gambar 4.19: Probabilitas pengelompokan dipisahkan secara linear dari N= r (*l*+ 1) titik dalam ruang *l* dimensi.

4.14 KLASIFIKASI POLINOMIAL

Pada bagian ini kita akan fokus pada salah satu kelas yang paling terkenal dari interpolasi fungsi $f_i(x)$ pada (4.47). Fungsi g(x) didekati sampai orde polinomial r dari komponen x, untuk r cukup besar. Untuk kasus khusus r = 2 kita mempunyai:

$$g(\boldsymbol{x}) = w_0 + \sum_{i=1}^{l} w_i x_i + \sum_{i=1}^{l-1} \sum_{m=i+1}^{l} w_{im} x_i x_m + \sum_{i=1}^{l} w_{ii} x_i^2 \qquad (4.51)$$

Jika , $\boldsymbol{x} = [x_1, x_2]^T$ kemudian bentuk umum y akan menjadi

$$\boldsymbol{y} = [x_1, x_2, x_1 x_2, x_1^2, x_2^2]^T$$

dan

$$g(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{y} + w_0$$

$$\boldsymbol{w}^T = [w_1, w_2, w_{12}, w_{11}, w_{22}]$$

Jumlah parameter bebas menentukan dimensi baru k. Generalisasi tersebut dari (4.51) untuk polinomial orde ke r sangat mudah, dan itu akan berisi hasil dari bentuk $x_1^{pl} x_2^{p2} \ldots x_l^{pl}$ dimana $p_1 + p_2 + \ldots + p_l \leq r$. Untuk sebuah tingkat polynomial ke-r dan x berdimensi-l dapat diperlihatkan sbb:

$$k = \frac{(l+r)!}{r!l!}$$

Untuk l = 10 dan r = 10 kita ambil k = 184,756 (!!). Yaitu, bahkan untuk harga ukuran medium dari tingkat jaringan dan kedimensian ruang masukan dari parameter bebas menjadi sangat tinggi. Mari kita perhatikan, sebagai contoh, masalah XOR terpisahkan-nonlinier. Tentukan

$$\boldsymbol{y} = \begin{bmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{bmatrix}$$

(4.52)

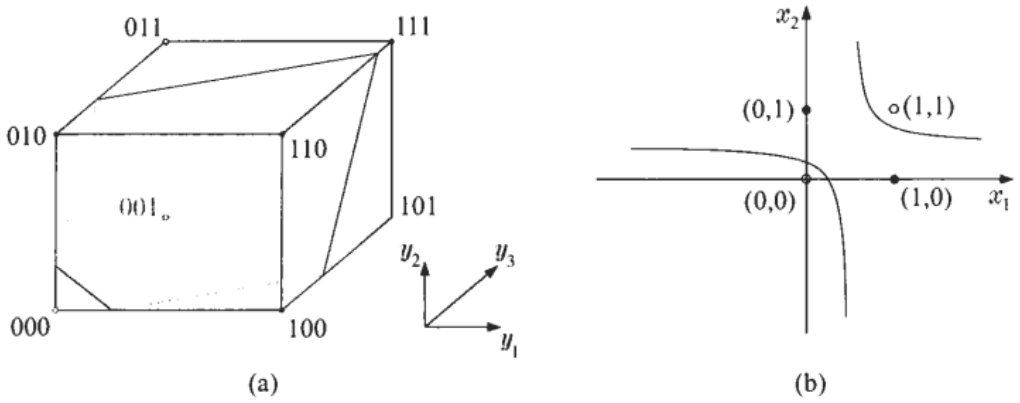
Vektor masukan dipetakan menjadi titik sudut dari sebuah unit dimensi-tiga (hyper) kubus, seperti yang ditunjukkan dalam gambar 4.20a ((00) → (000), (11) →(111), (10) → (100), (01) → (010). Titik-titik sudut ini dapat dipisahkan dengan bidang

$$y_1 + y_2 - 2y_3 - \frac{1}{4} = 0$$

Bidang dalam ruang dimensi-tiga ekuivalen dengan fungsi keputusan

$$g(\boldsymbol{x}) = -\frac{1}{4} + x_1 + x_2 - 2x_1 x_2 \begin{matrix} > 0 & \boldsymbol{x} \in A \\ < 0 & \boldsymbol{x} \in B \end{matrix}$$

Dalam ruang asli dimensi-dua, seperti yang ditunjukkan dalam gambar 4.20b.



Gambar 4.20: Tugas klasifikasi XOR, melalui pemilah linier polynomial tergeneralisir. (a) Bidang keputusan dalam ruang dimensi-tiga (b) Kurva keputusan dalam ruang asli dimensi-dua.

4.15 Jaringan Fungsi Dasar Radial

Fungsi interpolasi (kernel) yang akan dipertimbangkan dalam bagian ini adalah dari bentuk umum

$$f(\|\boldsymbol{x} - \boldsymbol{c}_i\|)$$

Artinya, argumen fungsi adalah jarak Euclidean dari vektor masukan x dari pusat ci, yang membenarkan nama fungsi basis radial (RBF). Fungsi f dapat berupa berbagai bentuk, misalnya,

$$f(\boldsymbol{x}) = \exp\left(-\frac{1}{2\sigma_i^2}\|\boldsymbol{x} - \boldsymbol{c}_i\|^2\right) \quad (4.53)$$

$$f(\boldsymbol{x}) = \frac{\sigma^2}{\sigma^2 + \|\boldsymbol{x} - \boldsymbol{c}_i\|^2} \quad (4.54)$$

Bentuk Gaussian lebih banyak digunakan. Untuk nilai yang cukup besar k, dapat ditunjukkan bahwa fungsi g (x) cukup didekati oleh [Broo 88, Mood 89]

$$g(\boldsymbol{x}) = w_0 + \sum_{i=1}^{k} w_i \exp\left(-\frac{(\boldsymbol{x} - \boldsymbol{c}_i)^T (\boldsymbol{x} - \boldsymbol{c}_i)}{2\sigma_i^2} \right) \qquad (4.55)$$

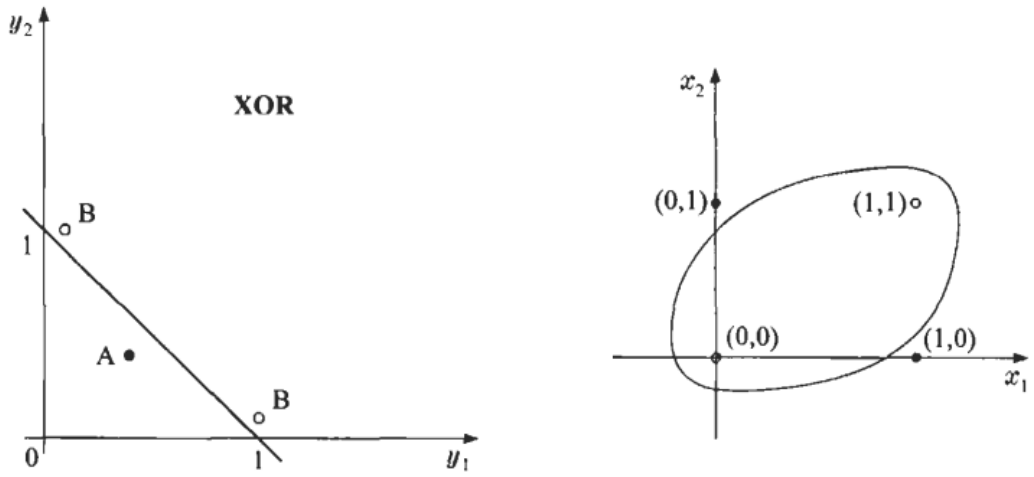
Artinya, pendekatan tersebut dicapai melalui penjumlahan RBF, di mana masing-masing terletak pada titik yang berbeda dalam ruang. Kita dapat dengan mudah mengamati bahwa ada hubungan yang erat antara hal ini dan pendekatan Parzen untuk fungsi probabilitas kepadatan pada Bab 2. Namun, perlu diketahui bahwa jumlah kernel yang dipilih untuk menjadi sama dengan jumlah k pelatihan poin = N. Sebaliknya, pada (4.55) k <<N. Selain keuntungan dalam kompleksitas komputasi, ini pengurangan jumlah kernel yang bermanfaat bagi kemampuan generalisasi dari model pendekatan yang dihasilkan.

Kembali ke Gambar 4.17, kita bisa menafsirkan (4.55) sebagai keluaran dari jaringan dengan satu lapisan tersembunyi dari fungsi aktivasi RBF (misalnya, (4.53), (4.54)) dan sebuah simpul keluaran linear. Sebagaimana telah disebutkan dalam Bagian 4.12, untuk sebuah masalah M-kelas akan ada M simpul keluaran linier. Pada titik ini, penting untuk menekankan satu perbedaan mendasar antara jaringan RBF dan perceptrons multi lapis. Selanjutnya, masukan untuk fungsi aktivas dari lapisan tersembunyi pertama, adalah kombinasi linier dari parameter masukan fitur ($\Sigma_i \omega_i x_i$;). Artinya, keluaran dari setiap neuron sama untuk semua (x: $\Sigma_i \omega_i x_i = c$), di mana c adalah suatu konstanta. Oleh karena itu, keluaran adalah sama untuk semua titik pada sebuah hyperplane Sebaliknya, dalam jaringan RBF keluaran setiap simpul RBF, fi (.) adalah sama untuk semua titik yang memiliki jarak Euclidean yang sama dari pusat masing-masing c_i; dan menurun secara eksponensial (untuk Gaussians) terhadap jarak. Dengan kata lain, respon aktivasi dari simpul dari alam lokal dalam RBF dan yang bersifat global dalam jaringan perceptron multi lapis. Perbedaan intrinsik memiliki dampak penting bagi kedua kecepatan konvergensi dan kinerja generalisasi Secara umum,. perceptron multi lapis belajar lebih lambat daripada rekan-rekan RBF mereka. Sebaliknya, perceptron multi lapis menunjukkan peningkatan sifat generalisasi, khususnya untuk daerah yang tidak cukup terwakili dalam himpunan pelatihan [Lane 91]. Hasil simulasi di Hart [90] menunjukkan bahwa, dalam rangka mencapai kinerja yang mirip dengan perceptrons multi lapis, jaringan RBF harus memiliki tingkat yang jauh lebih tinggi. Hal ini disebabkan lokalitas fungsi aktivasi RBF, yang membuatnya perlu untuk menggunakan sejumlah besar pusat untuk mengisi ruang di mana g (x) didefinisikan, dan jumlah ini menujukkan ketergantungan eksponensial pada dimensi dari ruang masukan (kutukan dimensi) [Hart 90].

Mari kita kembali kepada masalah XOR dan mengadopsi jaringan RBF untuk menyajikan pemetaan untuk masalah kelas linier terpisah. Pilih k = 2, pusat $\boldsymbol{c}_1 = [1, 1]^T, \boldsymbol{c}_2 = [0, 0]^T$ dan $f(\boldsymbol{x}) = \exp(-\|\boldsymbol{x} - \boldsymbol{c}_i\|^2)$. Y yang sesuai hasil pemetaan adalah

$$\boldsymbol{y} = \boldsymbol{y}(\boldsymbol{x}) = \begin{bmatrix} \exp(-\|\boldsymbol{x} - \boldsymbol{c}_1\|^2) \\ \exp(-\|\boldsymbol{x} - \boldsymbol{c}_2\|^2) \end{bmatrix}$$

Oleh karena (0,0) → (0.135, l), (1,1) → (1,0.135), (1,0) → (0.368,0.368), (0,1) → (0.368, 0.368). Gambar 4.21a menunjukkan posisi kelas yang dihasilkan setelah pemetaan di ruang y. Jelas, kedua kelas sekarang terpisah secara linear.



Gambar 4.21: Kurva keputusan dibentuk oleh pengklasifikasi linear RBF tergeneralisir untuk tugas XOR (a) dalam ruang terttransformasi dan (b) dalam ruang asli.

Dan garis lurus

$$g(\boldsymbol{y}) = y_1 + y_2 - 1 = 0$$

Merupakan suatu solusi yang mungkin. Gambar 4.21b menunjukkan kurva keputusan yang sama,

$$g(\boldsymbol{x}) = \exp(-\|\boldsymbol{x} - \boldsymbol{c}_1\|^2) + \exp(-\|\boldsymbol{x} - \boldsymbol{c}_2\|^2) - 1 = 0$$

Dalam ruang masukan vektor. Pada contoh kita, kita memilih pusat c_1, $_{c2}$ sebagai $[0,0]^T$ dan $[1,1]^T$. Pertanyaannya sekarang adalah, mengapa spesifik seperti ini? Ini adalah masalah penting untuk jaringan RBF. Beberapa petunjuk dasar tentang cara mengatasi masalah ini adalah sbb berikut ini.

Pusat Tetap

Walaupun telah ada beberapa kasus di mana sifat dari masalah menawarkan pilihan khusus untuk pusat [Theod 95], pada kasus umum pusat-pusat ini dapat dipilih secara acak dari himpunan pelatihan. Asalkan himpunan pelatihan didistribusikan secara merata pada semua ruang vektor fitur, hal ini tampaknya menjadi cara yang masuk akal untuk memilih pusat. Setelah dipilih pusat k untuk fungsi RBF, masalah telah berubah menjadi sebuah tipe linear pada ruang berdimensi-k dari vektor y,

$$\boldsymbol{y} = \begin{bmatrix} \exp\left(\frac{-\|\boldsymbol{x}-\boldsymbol{c}_1\|^2}{2\sigma_1^2}\right) \\ \vdots \\ \exp\left(\frac{-\|\boldsymbol{x}-\boldsymbol{c}_k\|^2}{2\sigma_k^2}\right) \end{bmatrix}$$

dimana varians juga dipertimbangkan untuk diketahui, dan

$$g(\boldsymbol{x}) = w_0 + \boldsymbol{w}^T \boldsymbol{y}$$

Semua metode dipaparkan dalam Bab 3, sekarang dapat digunakan untuk memperkirakan ω_0 dan ω.

Pelatihan Pusat

Jika pusat tidak dipilih sebelumnya, mereka harus diperkirakan selama fase pelatihan dengan beban ω_i dan varians σ_i^2, jika yang terakhir ini juga dianggap tidak diketahui. N adalah jumlah masukan dari pasangan keluaran yang diinginkan (x (j), y (j), j = 1,..., N). Kita pilih fungsi biaya yang tepat dari error keluaran

$$J = \sum_{j=1}^{N} \phi(e(j))$$

Dimana ϕ(.) merupakan sebuah fungsi yang errornya dapat diferensialkan (contoh, akar dari argumentnya).

Estimasi bobot ω_i, pusat c_i, dan varians σ_i^2 menjadi tugas khusus dari proses optimasi nonlinier. Sebagai contoh, jika kita mengadopsi pendekatan turunan gradient, algoritma menghasilkan hasil sebagai berikut:

$$w_i(t+1) = w_i(t) - \mu_1 \left.\frac{\partial J}{\partial w_i}\right|_t, \quad i = 0, 1, \ldots, k \qquad (4.56)$$

$$\boldsymbol{c}_i(t+1) = \boldsymbol{c}_i(t) - \mu_2 \left.\frac{\partial J}{\partial \boldsymbol{c}_i}\right|_t, \quad i = 1, 2, \ldots, k \qquad (4.57)$$

$$\sigma_i(t+1) = \sigma_i(t) - \mu_3 \left.\frac{\partial J}{\partial \sigma_i}\right|_t, \quad i = 1, 2, \ldots, k \qquad (4.58)$$

dimana t adalah langkah iterasi saat ini. Kompleksitas komputasi seperti skema adalah penghalang untuk sejumlah situasi praktis. Untuk mengatasi kekurangan ini, berbagai tehnik alternatif diusulkan.

Salah satu cara adalah dengan memilih pusat dalam cara perwakilan atas jalur data yang didistribusikan dalam ruang. Hal ini dapat dicapai dengan mengungkap sifat pengelompokan data dan memilih wakil yang representative untuk setiap cluster [Mood 89]. Hal ini merupakan masalah khas dari pembelajaran tanpa pengawasan (*unsupervised learning*) dan algoritma yang dibahas dalam bab-bab selanjutnya dalam buku ini dapat digunakan. Bobot tak diketahui ωi, kemudian belajar melalui skema terawasi (misalnya, algoritma gradient turun) untuk meminimalkan error keluaran. Dengan demikian, skema menggunakan kombinasi prosedur pembelajaran terawasi dan tanpa pengawasan.

Suatu strategi alternatif dijelaskan dalam [Chen 91]. Sejumlah besar calon pusat awalnya dipilih dari sekumpulan pelatihan vektor. Kemudian, sebuah teknik regresi linier maju digunakan, seperti kuadrat terkecil ortogonal, yang mengarah ke sekumpulan pelatihan pusat yang sedikit. Teknik ini juga menyediakan cara untuk memperkirakan urutan model k. Suatu bentuk rekursif dari metode, yang mampu unggul dalam penghematan komputasi, diberikan dalam [Gomm 00]. Baru-baru ini, metode lain telah diusulkan berdasar dukungan mesin vektor. Ide dibalik metodologi ini adalah untuk melihat jaringan RBF sebagai sebuah mesin pemetaan, melalui kernel, ke suatu ruang berdimensi tinggi. Kemudian kami merancang sebuah pemilah hyperplane dengan menggunakan vektor yang paling dekat dengan batas keputusan.

Hal ini adalah vektor dukungan dan sesuai dengan pusat-pusat ruang masukan. Pelatihan ini terdiri dari masalah pemrograman kuadratik dan menjamin global optimum [Scho 97]. Fitur

bagus dari algoritma ini adalah bahwa ia secara otomatis menghitung semua parameter yang tidak diketahui termasuk jumlah pusat. Kita akan kembali lagi nanti dalam bab ini.

Dalam Plat[91] sebuah pendekatan yang serupa semangatnya dengan teknik konstruktif, didiskusikan untuk perceptrons multi lapis , telah disarankan. Idenya adalah untuk memulai pelatihan jaringan RBF dengan beberapa simpul (awalnya satu) dan terus berkembang jaringan dengan mengalokasikan yang baru, berdasarkan pada "hal baru" dalam vektor fitur yang datang secara berurutan. Kebaruan dari setiap pelatihan yang diinginkan pasangan masukan-keluaran ditentukan oleh dua kondisi: a) vektor masukan menjadi sangat jauh (berdasarkan pada ambang) dari semua pusat yang sudah ada dan b) error keluaran yang sesuai (menggunakan jaringan RBF dilatih sampai saat ini) lebih besar dari yang lain yang telah ditetapkan ambang batasnya. Jika kedua kondisi terpenuhi maka vektor masukan baru ditugaskan sebagai pusat baru. Jika tidak, pasangan masukan-keluaran yang diinginkan yang digunakan untuk meng-*update* parameter jaringan sesuai dengan algoritma pelatihan yang diadopsi, misalnya skema gradient menurun. Suatu varian dari skema yang memungkinkan pemindahan pusat sebelumnya juga telah disarankan dalam [Ying 98]. Hal ini pada dasarnya merupakan kombinasi dari filosofi konstruktif dan pemangkasan. Prosedur yang disarankan pada Kara[97] juga bergerak sepanjang arah yang sama. Namun, tugas dari pusat baru didasarkan pada prosedur pemilahan progresif (sesuai dengan kriteria pemilahan) dari ruang fitur dengan menggunakan pengelompokan atau teknik kuantisasi vector pembelajaran (Bab 14). Para anggota dari daerah yang telah ditetapkan sebagai pusat dari RBF's. Seperti halnya dengan teknik tersebut. pertumbuhan dan pelatihan dilakukan secara bersamaan. Sejumlah teknik lainnya juga telah diusulkan. Untuk review lihat, misalnya, [Hush 93]. Sebuah perbandingan jaringan RBF dengan strategi pemilihan pusat yang berbeda versus perceptrons multi lapis dalam konteks pengenalan tutur seperti yang disajikan dalam [Wett 92]. Tinjauan melibatkan jaringan RBF dan aplikasi terkaitnya seperti disajikan dalam [Hayk 96, Mulg 96].

4.16 APPROXIMATOR UMUM

Pada bagian ini kami memberikan panduan dasar tentang pendekatan sifat fungsi nonlinier yang digunakan di seluruh bab ini, yaitu, sigmoid. polinomial, dan fungsi dasar radial. Teorema yang dinyatakan membenarkan penggunaan jaringan yang sesuai sebagai approximators permukaan putusan serta approximator fungsi probabilitas, tergantung pada bagaimana kita melihat pemilah. Dalam (4,51) perluasan polinomial digunakan untuk perkiraan fungsi nonlinier g(x). Pilihan ini sebagai fungsi pendekatan telah dibenarkan oleh Teorema Weierstrass.

Teorema. Dengan g (x) menjadi fungsi kontinu yang dinyatakan dalam sebuah himpunan bagian (tertutup) kompak S C R ', dan ε> 0. Kemudian terdapat sebuah integer r = r (ε) dan sebuah fungsi polinomial Φ(x) dari tingkat- r, sehingga

$$|g(\boldsymbol{x}) - \boldsymbol{\phi}(\boldsymbol{x})| < \epsilon, \quad \forall \boldsymbol{x} \in S$$

Dengan kata lain, fungsi g (x) dapat didekati secara dekat dengan r cukup besar. Masalah utama yang terkait dengan ekspansi polinomial adalah bahwa perkiraan yang baik biasanya dicapai untuk nilai r yang besar. Artinya, konvergensi untuk g (x) lambat. Dalam [Barr 93] terlihat bahwa error pendekatan dikurangi sesuai dengan aturan O ($\frac{1}{r^{2/l}}$), dimana O (.) menunjukkan urutan magnitude. Dengan demikian, error berkurang secara perlahan dengan meningkatnya dimensi-l ruang masukan, dan nilai-nilai besar r diperlukan untuk error

pendekatan yang diberikan. Namun, nilai besar r, selain kompleksitas komputasi dan isu-isu generalisasi (karena banyaknya parameter bebas yang diperlukan), juga menyebabkan perilaku akurasi numerik yang memprihatinkan dalam perhitungan, karena jumlah banyaknya produk yang terlibat. Di sisi lain, ekspansi polynomial dapat dipergunakan secara efektif untuk pendekatan sepenggal demi sepenggal, dimana r yang lebih kecil bisa diadopsi. Penurunan lambat dari error pendekatan terhadap tatanan sistem dan dimensi ruang masukan yang umum bagi semua perluasan dari bentuk (4,47) dengan fungsi basis dasar fi(.) tetap. Skenario menjadi berbeda jika data fungsi adaptif dipilih, seperti halnya dengan perceptrons multi lapis. Di kedua, argumen dalam fungsi aktivasi adalah f ($\omega^T x$), dengan ω dihitung secara optimal dari data yang tersedia.
Mari kita pertimbangkan perceptron dua lapis dengan satu lapisan tersembunyi, memiliki k simpul dengan fungsi aktivasi f(.) dan simpul keluaran dengan aktivasi linear. Keluaran dari jaringan ini kemudian diberikan oleh

$$\phi(\boldsymbol{x}) = \sum_{j=1}^{k} w_j^o f(\boldsymbol{w}_j^{hT} \boldsymbol{x}) + w_o^o \tag{4.59}$$

dimana h mengacu pada bobot, termasuk ambang batas dari lapisan tersembunyi dan o pada bobot lapisan keluaran. Diperoleh bahwa f (.) adalah fungsi *squashing*, teorema berikut menetapkan sifat pendekatan umum seperti pada suatu jaringan [Cybe 89, Funa 89, Horn 89, Ito 91, Kalo 97].

Teorema. Dengan g(x) merupakan fungsi kontinu yang didefinisikan dalam himpunan bagian kompak S C R' dan ε> 0. Kemudian terdapat k= k (ε) dan sebuah perceptron dua-lapis (4.59) sehingga

$$|g(\boldsymbol{x}) - \phi(\boldsymbol{x})| < \epsilon, \quad \forall \boldsymbol{x} \in S$$

Dalam [Barr 93], terlihat bahwa, berbeda dengan perluasan polinomial, error pendekatan menurun sesuai dengan aturan O ($\frac{1}{k}$). Dengan kata lain, ruang masukan dimensi tidak masuk secara eksplisit ke tempat dan error berbanding terbalik terhadap tingkat sistem, yaitu jumlah neuron. Jelas, harga yang kita bayar untuk itu adalah bahwa proses optimasi sekarang nonlinier, dengan kekurangan terkait pada potensi untuk konvergensi menuju minimum lokal. Pertanyaan yang sekarang muncul adalah apakah kita mendapatkan apa-apa dengan menggunakan lebih dari satu lapisan tersembunyi, saat satu lapisanpun sudah cukup untuk pendekatan fungsi. Jawabannya adalah bahwa menggunakan lebih dari satu lapisan dapat menyebabkan pendekatan yang lebih efisien, yaitu, akurasi yang sama dicapai dengan neuron lebih sedikit dalam jaringan. Properti pendekatan umum ini juga berlaku untuk kelas fungsi RBF. Untuk nilai yang cukup besar k dalam (4.55) perluasan yang dihasilkan dapat mendekati secara dekat setiap fungsi kontinu dalam sebuah himpunan bagian kompak S [Park 91, Park 93].

4.17 MESIN VEKTOR PENDUKUNG: KASUS NONLINIER

Dalam Bab 3, kita membahas mesin vektor dukungan (SVM) sebagai suatu metodologi desain yang optimal dari pemilah linear. Mari kita berasumsi bahwa terdapat pemetaan

$$\boldsymbol{x} \in \mathcal{R}^l \longrightarrow \boldsymbol{y} \in \mathcal{R}^k$$

dari ruang masukan fitur ke suatu ruang berdimensi-k, dimana kelas secara memuaskan dapat dipisahkan oleh sebuah hyperplane. Kemudian, dalam rangka dibahas dalam Bagian 4.12, metode SVM dapat dimobilisasi untuk desain pemilah hyperplane dalam ruang berdimensi-k baru. Namun, terdapat properti yang elegan dalam metodologi SVM, yang dapat dimanfaatkan untuk pengembangan pendekatan yang lebih umum. Ini juga akan memungkinkan kita untuk pemetaan (implisit) dalam ruang berdimensi tak terbatas, jika diperlukan.
Ingat dari Bab 3 bahwa, dalam perhitungan yang terlibat dalam dual representasi Wolfe, vektor fitur berpartisipasi dalam pasangan, melalui operasi hasil kali kedalam. Juga, setelah hyperplane optimal (ω,ω_0) telah dihitung, klasifikasi yang dilakukan berdasarkan apakah tanda

$$\begin{aligned} g(\boldsymbol{x}) &= \boldsymbol{w}^T \boldsymbol{x} + w_0 \\ &= \sum_{i=1}^{N_s} \lambda_i y_i \boldsymbol{x}_i^T \boldsymbol{x} + w_0 \end{aligned}$$

+ atau -, dimana N_s, merupakan jumlah vektor dukungan. Jadi, sekali lagi, hanya produk dalam yang masuk ke tempat. Jika desain adalah untuk mengambil tempat di ruang berdimensi-k baru, satu-satunya perbedaan adalah bahwa vektor yang terlibat akan menjadi pemetaan dimensi-k dari vektor fitur masukan asli. Tampak naif pada hal itu akan menyebabkan kesimpulan bahwa kompleksitas sekarang jauh lebih tinggi, karena, biasanya k jauh lebih tinggi dari dimensi ruang masukan l, untuk membuat kelas linier terpisah. Namun demikian, ada kejutan bagus yang hanya menunggu kita. Mari kita mulai dengan contoh sederhana. Asumsikan bahwa

$$\boldsymbol{x} \in \mathcal{R}^2 \longrightarrow \boldsymbol{y} = \begin{bmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix}$$

Kemudian, ini adalah masalah aljabar sederhana untuk menunjukkan bahwa

$$\boldsymbol{y}_i^T \boldsymbol{y}_j = \left(\boldsymbol{x}_i^T \boldsymbol{x}_j\right)^2$$

Dengan kata lain, *inner product* dari vektor dalam ruang baru (berdimensi lebih tinggi) telah dinyatakan sebagai suatu fungsi dari produk bagian dalam vektor yang sesuai di ruang fitur asli. Paling menarik!

Teorema. Mercer Teorema. Biarkan x E R'dan suatu pemetaan Φ

$$\boldsymbol{x} \longrightarrow \boldsymbol{\phi}(\boldsymbol{x}) \in H$$

dimana H merupakan suatu ruang Euclidean. Kemudian operasi *inner product* memiliki sebuah representasi setara

$$\sum_r \phi_r(\boldsymbol{x})\phi_r(\boldsymbol{z}) = K(\boldsymbol{x}, \boldsymbol{z})$$

dimana $\Phi_r(x)$ adalah komponen-r dari pemetaan $\Phi(x)$ dari x, dan K (x, z) merupakan sebuah fungsi simetris yang memenuhi kondisi berikut

$$\int K(\boldsymbol{x}, z) g(\boldsymbol{x}) g(z)\, d\boldsymbol{x}\, dz \geq 0$$

untuk setiap g (x), x E R' sehingga

$$\int g(\boldsymbol{x})^2\, d\boldsymbol{x} < +\infty$$

sebaliknya juga benar,seperti untuk setiap fungsi K(x, z) memenuhi (4.61) dan (4.62) terdapatsuatu ruang dimana K (x, z) mendefinisikansuatu *inner product*! Fungsi seperti ini juga dikenal sebagai kernel. Apa, bagaimanapun, teorema Mercer tidak mengungkapkan kepada kita adalah bagaimana menemukan ruang ini. Artinya, kita tidak memiliki alat umum untuk membangun pemetaan $\Phi(.)$ begitu kita mengetahui *inner product* ruang yang sesuai. Selanjutnya, kami tidak memiliki sarana untuk mengetahui dimensi ruang, yang bahkan bisa tak terbatas. Untuk lebih lanjut tentang isu-isu ini, secara matematis pembaca dapat mempelajari[Cour 53]. Contoh umum kernel yang digunakan dalam aplikasi pengenalan pola adalah

Polinomial

$$K(\boldsymbol{x}, z) = (\boldsymbol{x}^T z + 1)^q, \quad q > 0 \tag{4.63}$$

Radial Basis Function

$$K(\boldsymbol{x}, z) = \exp\left(-\frac{\|\boldsymbol{x} - z\|^2}{\sigma^2}\right) \tag{4.64}$$

Tangen Hiperbola

$$K(\boldsymbol{x}, z) = \tanh\left(\beta \boldsymbol{x}^T z + \gamma\right) \tag{4.65}$$

untuk nilai β dan γ yang sesuai sehingga kondisi Mercer terpenuhi. Salah satu kemungkinannya adalah $\beta = 2$, dan $\gamma = 1$.

Setelah sebuah kernel yang diperlukan telah diadopsi bahwa secara implisit mendefinisikan pemetaan ke dalam ruang berdimensi yang lebih tinggi, tugas optimasi dual Wolfe(Persamaan. (3.91) -(3.93)) menjadi

$$\max_{\boldsymbol{\lambda}} \left(\sum_i \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \right) \tag{4.66}$$

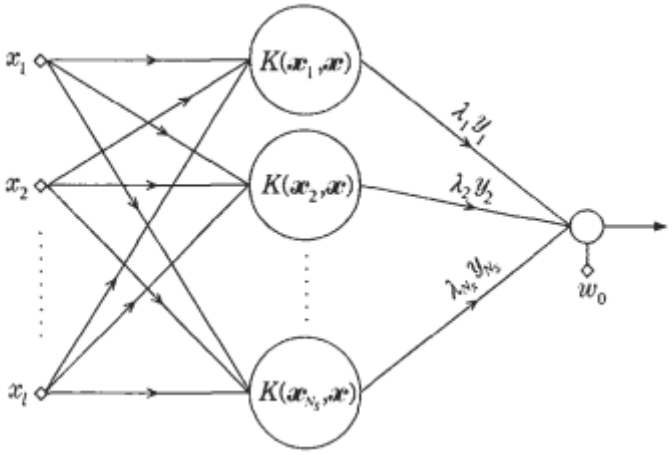
$$\text{subject to} \quad 0 \leq \lambda_i \leq C, \quad i = 1, 2, \ldots, N \tag{4.67}$$

$$\sum_i \lambda_i y_i = 0 \tag{4.68}$$

dan pemilahan linier yang dihasilkan adalah sbb:

$$\text{assign } \boldsymbol{x} \text{ in } \omega_1(\omega_2) \text{ if } g(\boldsymbol{x}) = \sum_{i=1}^{N_s} \lambda_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + w_0 > (<)\, 0 \tag{4.69}$$

Gambar 4.22 menunjukkan arsitektur yang sesuai. Ini tidak lain dari sebuah kasus khusus dari pemilah lineartergeneralisasi pada Gambar 4.17. Jumlah simpul ditentukan oleh jumlah vektor dukungan N_s, Simpul melakukan *inner product* antara pemetaan dari x dan pemetaan yang berkaitan dengan vektor dukungan dalam ruang berdimensi tinggi, melalui operasi kernel



Gambar 4.22**:** Arsitektur SVM menggunakan fungsi kernel

Keterangan

- Perhatikan bahwa jika fungsi kernel adalah RBF, maka arsitekturnya adalah sama dengan arsitektur jaringan RBF Gambar 4.17. Namun, pendekatan yang diikuti di sini berbeda. Dalam Bagian 4.15, suatu pemetaan dalam ruang berdimensi-k pertama kali dilakukan dan pusat fungsi RBF harus diperkirakan. Dalam pendekatan SVM, jumlah simpul serta pusat-pusat adalah hasil dari prosedur optimasi.
- Fungsi tangen hiperbolik merupakan suatu sigmoid. Jika dipilih sebagai sebuah kernel, arsitektur yang dihasilkan merupakan suatu kasus khusus dari perceptron lapis dua. Terlebih lagi, jumlah simpul merupakan hasil dari prosedur optimasi. Hal ini penting. Meskipun arsitektur SVM adalah sama dengan yang dari perceptron dua lapis, prosedur pelatihan yang sama sekali berbeda untuk kedua metode. Hal yang sama berlaku untuk jaringan RBF.
- Karakteristik penting dari mesin vektor dukungan adalah bahwa kompleksitas perhitungan independen terhadap dimensi ruang kernel, dimana ruang fitur masukan dipetakan. Jadi, kutukan dimensi terlewati. Dengan kata lain, satu desain dalam ruang dimensi tinggi tanpa harus mengadopsi model eksplisit dengan menggunakan sejumlah besar parameter, karena hal ini akan ditentukan oleh dimensi ruang yang tinggi. Ini juga berpengaruh pada sifat generalisasi dan memang SVM's cenderung menunjukkan unjuk kerja generalisasi yang baik. Kami akan kembali ke isu ini pada akhir Bab 5.
- Keterbatasan utama dari mesin vektor dukungan adalah tingginya beban perhitungan yang diperlukan, baik selama pelatihan dan dalam tahap uji coba. Untuk masalah dengan sejumlah data pelatihan yang relative kecil, setiap algoritma optimasi tujuan

umum dapat digunakan. Namun demikian, untuk sejumlah pelatihan yang besar (ukuran beberapa ribu), diperlukan suatu perlakuan khusus. Pelatihan SVM biasanya dilakukan dalam mode batch. Untuk masalah besar ini menyebabkan permintaan yang tinggi atas kebutuhan memori komputer. Untuk menyerang proabilitas sejumlah prosedur telah dibuat. Filosofinya bergantung pada dekomposisi, pada satu cara atau cara lainnya, dari optimasi masalah ke suatu rangkaian yang lebih kecil, misalnya, [Bose 92, Osun 97, Chan 00]. Baru-baru ini, sebuah algoritma telah diajukan untuk menyelesaikan masalah secara iteratif, dengan asumsi bahwa pada setiap langkah iterasi hanya dua dari pengali Langrange yang tidak diketahui dan sisanya diketahui (dimulai dengan asumsi awal). Prosedur mengambil keuntungan dari fakta bahwa suatu solusi analisis untuk kedua masalah adalah mungkin untuk kasus dua pengali Langrange [Matt 99, Plat 99]. Algoritma sekuensial yang lain telah diusulkan dalam [Navi 01], di mana sebuah prosedur iteratif kuadrat terkecil terbobot-ulang dipergunakan dan alternatif optimasi berat dengan kendala memaksa. Sebuah keuntungan dari teknik yang terakhir ini bahwa hal ini secara alami mengarah ke implementasi *online* dan adaptif.

Untuk masalah besar, tahap uji coba juga bisa sangat menuntut, jika jumlah vektor dukungan terlalu tinggi. Metode yang mempercepat perhitungan juga telah diusulkan, misalnya [Burg 97].

- Keterbatasan utama lainnya dari mesin vektor dukungan adalah bahwa, sampai sekarang. tidak ada metode praktis untuk pilihan terbaik dari fungsi kernel. Hal ini masih merupakan suatu masalah penelitian namun menantang, masalah dalam penelitian. Mesin vector dukungan telah diterapkan ke sejumlah aplikasi beragam, mulai dari pengenalan tulisan tangan ([Cort 95]), untuk pengenalan objek ([Blan 96]), identifikasi orang ([Ben 99]), kategorisasi spam ([Druc 99]), dan pemerataan kanal ([Seba 001]). Hasil dari aplikasi ini menunjukkan bahwa pemilah SVM menunjukkan peningkatan kinerja generalisasi, yang tampaknya menjadi kekuatan mesin vektor dukungan.
- Selain mesin vektor dukungan setiap pemilah linier lain yang menggunakan *inner product* dapat secara implisit dilaksanakan dalam ruang-ruang berdimensi yang lebih tinggi dengan menggunakan kernel. Dengan cara yang satu ini dapat secara elegan membangun versi nonlinier dari suatu algoritma linier. Untuk meninjau mengenai isu ini lihat, misalnya. [Mull 01].

4.18 POHON KEPUTUSAN (DECISION TREE)

Pada bagian ini kita meninjau secara singkat kelas besar dari pemilah nonlinier yang dikenal sebagai pohon keputusan. Pohon keputusan merupakan sistem keputusan multi-langkah, di mana kelas-kelas secara berurutan ditolak sampai kita mencapai kelas yang akhirnya diterima. Untuk tujuan ini, ruang fitur dibagi menjadi daerah yang unik, sesuai dengan kelas, secara berurutan.

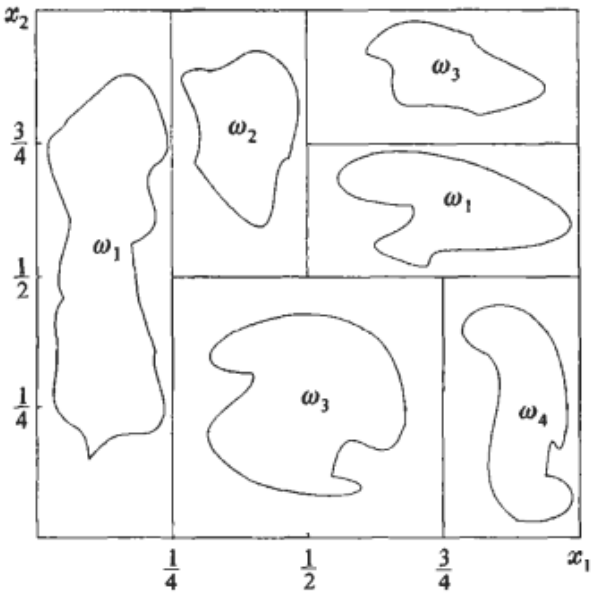
Setelah kedatangan suatu vektor fitur, dengan mencari daerah dimana vektor fitur akan ditugaskan dapat dicapai melalui urutan keputusan sepanjang jalur simpul dari pohon yang secara tepat dibangun. Skema tersebut menawarkan berbagai keuntungan ketika sejumlah besar kelas dilibatkan. Yang paling populer di antara pohon-pohon keputusan adalah mereka yang membagi ruang menjadi hyperrectangles dengan sisi sejajar dengan sumbu. Urutan keputusan diterapkan pada fitur individu, dan pertanyaan-pertanyaan yang harus dijawab dari " apakah fitur $x_i \leq \alpha$? "dimana α adalah nilai ambang. Pohon tersebut dikenal sebagai *Ordinary Binary Classification Trees* (OBCTs). Jenis pohon lainnya juga mungkin,

membagi ruang ke dalam sel polyhedral cembung atau menjadi potongan-potongan dari lingkungan.

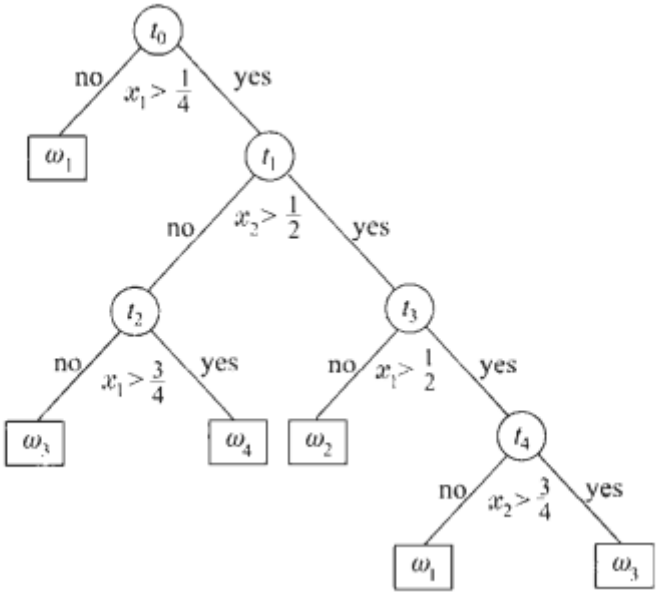
Ide dasar dibalik OBCT adalah ditunjukkan melalui contoh sederhana Gambar 4.23. Dengan pemilahan berurutan berturut-turut ruang yang kita telah menciptakan daerah sesuai dengan berbagai kelas.

Gambar 4.24 menunjukkan masing-masing pohon biner dengan simpul keputusan dan daun. Perhatikan bahwa adalah mungkin untuk mencapai keputusan tanpa menguji semua fitur yang tersedia.

Tugas diilustrasikan pada Gambar 4.23 adalah salah satu yang sederhana dalam ruang dua dimensi. Ambang batas yang digunakan untuk binary split pada setiap simpul dari pohon pada Gambar 4.24 di ungkapkan dengan pengamatan sederhana dari masalah geometri. Namun, hal ini tidak mungkin dalam ruang dimensi yang lebih tinggi. Selanjutnya, kami mulai query dengan menguji x_I terhadap 1/4. Sebuah pertanyaan yang jelas adalah mengapa untuk mempertimbangkan x_1 pertama dan tidak fitur lain. Dalam kasus umum, dalam rangka untuk mengembangkan pohon keputusan biner,



Gambar 4.23: Keputusan partisi pohon



elemen desain berikut harus dipertimbangkan oleh perancang dalam tahap pelatihan:

- Pada setiap simpul, sekumpulan pertanyaan calon yang diminta harus diputuskan. Setiap pertanyaan berkaitan dengan suatu pemecahan biner tertentu menjadi dua simpul *descendant*. Setiap simpul, t, terkait dengan suatu himpunan bagian x_t tertentu dari himpunan pelatihan x. Pemilahan sebuah simpul setara dengan pemilahan himpunan bagian x_t menjadi dua himpunan bagian beririsan keturunan, x_{tY}, $x_{tN.}$ Yang pertama dari dua terdiri dari vektor di x_t yang sesuai dengan jawaban "Ya" dari pertanyaan dan jawaban “Tidak” dari yang kedua. Yang pertama (akar) dari pohon dikaitkan dengan pelatihan himpunan x. Untuk setiap pemilahan, berikut ini benar:

$$X_{tY} \bigcap X_{tN} = \emptyset$$

$$X_{tY} \bigcup X_{tN} = X_t$$

- Sebuah kriteria pemilahan harus diambil sesuai dengan pemilahan terbaik dari himpunan kandidat yang terpilih.
- Aturan stop-pemilahan diperlukan untuk mengontrol pertumbuhan pohon dan simpul dinyatakan sebagai salah satu terminal (daun.)
- Suatu aturan yang dibutuhkan sehingga setiap daun ditetapkan untuk kelas tertentu.

Kita sekarang cukup berpengalaman untuk mengharapkan bahwa ada lebih dari satu metode untuk mendekati setiap elemen desain di atas.

4.18.1 Himpunan Pertanyaan

Untuk jenis pohon OBCT pertanyaan adalah dalam bentuk "Apakah $x_k \leq \alpha$?" Untuk masing-masing fitur, setiap nilai yang mungkin dari ambang pintu α mendefinisikan pemecahan tertentu himpunan bagian X_t. Jadi dalam teori, sebuah himpunan tak terhingga dari pertanyaan yang harus ditanyakan apakah α bervariasi dalam interval $Y_\alpha \subseteq R$. Dalam prakteknya, hanya satu himpunan pertanyaan berhingga yang dapat dipertimbangkan. Sebagai contoh, karena jumlah, N, titik pelatihan di X adalah berhingga, salah satu fitur x_k, k = 1,. . . , 1, dapat mengambil paling $N_t \leq N$ nilai yang berbeda, di mana N_t adalah kardinalitas dari himpunan bagian $X_t \subseteq X$. Dengan demikian, untuk fitur x_k, kita dapat menggunakan α_{kn}, n = 1,2,. . . . N_{tk} ($N_{tk} \leq Nt$), dimana α_{kn} diambil pertengahan antara nilai-nilai yang berbeda

secara berturut-turut pada himpunan bagian x_k pelatihan X_t. Hal yang sama harus diulang lagi untuk semua fitur. Jadi dalam kasus seperti itu, jumlah pertanyaan calon$\sum_{k=1}^{i} Ntk$. Namun, hanya salah satu dari mereka yang harus dipilih untuk memberikan pemilahanan biner di simpul saat ini, t, dari pohon. Ini dipilih untuk mengarahahkan ke pemilahan yang terbaik yang terkait dengan himpunan bagian X_r. Pemilahan terbaik adalah berdasarkan pada kriteria pemilahan.

4.18.2 Kriteria Pemilahan

Setiap pemilahan biner dari suatu simpul t, menghasilkan dua simpul keturunan. Mari kita nyatakan mereka oleh t_Y dan t_N sesuai dengan jawaban "Ya" atau "Tidak" pada pertanyaan tunggal yang diambil untuk t simpul, juga disebut sebagai simpul leluhur. Seperti yang telah kita sebutkan, simpul keturunan dihubungan dengan dua himpunan bagian baru, yaitu masing-masing X_{tY}, X_{tN}. Agar metodologi pohon tumbuh, dari simpul akar ke simpul daun, agar masuk akal, setiap pemilahan harus menghasilkan himpunan bagian yang lebih "kelas homogen" dibanding X_t himpunan bagian nenek moyang. Ini berarti bahwa fitur vektor pelatihan di masing-masing himpunan bagian baru menunjukkan preferensi yang lebih tinggi untuk kelas yang spesifik, sedangkan data dalam X_t lebih merata terdistribusi diantara kelas. Sebagai contoh, mari kita perhatikan tugas empat-kelas dan mengasumsikan bahwa vektor di himpunan bagian X_t didistribusikan di antara kelas dengan probabilitas yang sama (persentase). Jika salah satu simpul membelah sehingga poin yang milik kelas ω_1, ω_2 membentuk himpunan bagian X_{tY}, dan poin dari ω_3, ω_4 kelas membentuk himpunan bagian X_{tN}, maka himpunan bagian baru lebih homogen dibandingkan dengan X_t, atau "lebih murni" dalam terminologi keputusan pohon. Tujuannya adalah untuk menentukan ukuran yang mengkuantifikasi ketidak-murnian dan membagi simpul simpul sehingga ketidak-murnian secara keseluruhan dari simpul keturunan optimal menurun sehubungan dengan ketidak-murnian simpul nenek moyang. Misalkan P (ω_i|t) menunjukkan probabilitas bahwa vektor di himpunan bagian X_t, dikaitkan dengan simpul t, ditunjukkan sebagai kelas ω_i, i= 1,2,. . . , M. yang biasa digunakan untuk mendifinisikan ketidakmurnian simpul, dinotasikan sebagai I (t), sbb:

$$I(t) = -\sum_{i=1}^{M} P(\omega_i|t) \log_2 P(\omega_i|t)$$

dimana $\log_2$ adalah logaritma dengan basis 2. Ini tidak lain dari entropi yang terkait dengan himpunan bagian Xt, yang dikenal dari Teori Informasi Shannon. Hal ini tidak sulit untuk menunjukkan bahwa I (t) membutuhkan nilai maksimum jika semua probabilitas sama dengan $\frac{1}{M}$ (ketidak-murnian tertinggi) dan menjadi nol (ingat bahwa 0log0 = 0) jika semua data termasuk ke dalam kelas tunggal, yaitu, jika hanya salah satu P(ω_i|t) = 1 dan semua yang lain adalah nol (ketidak-murnian minimal). Dalam prakteknya, prosentarse probabilitas$\frac{Ni}{Nt}$ dimana N_t^i merupakan jumlah titik dalam Xt yang termasuk dalam kelas ω_i. Asumsikan sekarang bahwa melakukan pemilahan, poin N_{tY} dikirimkan ke simpul "Ya" (X_{tY}) dan ke simpul "Tidak" (X_{tN}). Penurunan pengotor simpul didefinisikan sebagai

$$\Delta I(t) = I(t) - \frac{N_{tY}}{N_t} I(t_Y) - \frac{N_{tN}}{N_t} I(t_N)$$

di mana I(tY), Z (tN) adalah ketidak-murnian dari simpul tY dan tN masing-masing. Tujuannya sekarang menjadi untuk mengadopsi, dari himpunan pertanyaan kandidat, salah satu yang melakukan pemilahan mengarah pada penurunan ketidak-murnian tertinggi.

4.18.3 Aturan Stop-Pemilahan

Pertanyaan alami yang sekarang muncul adalah ketika seseorang memutuskan untuk berhenti memisahkan simpul dan menyatakannya sebagai daun pohon. Sebuah kemungkinan untuk mengadopsi ambang batas T dan menghentikan pemilahan jika nilai maksimum $\Delta I(t)$, atas segala kemungkinan pemilahan, kurang dari T. Alternatif lain untuk menghentikan pemilahan baik jika kardinalitas himpunan bagian X_t cukup kecil atau jika X_t murni, dalam pengertian bahwa semua titik di dalamnya milik satu kelas.

4.18.4 Aturan Penugasan Kelas

Sekali sebuah simpul dinyatakan menjadi daun, maka harus diberi label kelas. Aturan umum yang digunakan adalah aturan mayoritas, seperti daun dilabeli sebagai ω_j dimana

$$j = arg \max_i P(\omega_i|t)$$

Dengan kata-kata dapat dinyatakan, kita tetapkan sebuah daun, t, ke kelas yang vector mayoritasnya dimiliki X_t. Setelah membahas elemen utama yang diperlukan untuk pertumbuhan pohon keputusan, kita sekarang siap untuk meringkas langkah-langkah algoritma dasar untuk membangun pohon keputusan biner

- Mulailah dengan simpul akar, seperti, $X_t = X$.
- Untuk setiap t simpul baru
  * Untuk setiap fitur x_k, k = 1,2,. . . , 1
    # Untuk setiap nilai α_{kn}, n = 1,2,. . . , N_{tk}
      ~ Bangkitkan X_{tY} dan X_{tN} sesuai dengan jawaban pada pertanyaan: apakah $x_k(i) \leq \alpha_{kn}$, i= 1,2,…N_t
      ~ Hitung penurunan ketidak-murnian
    # END
    #Pilih x_{k0} dan α_{k0n0} mengarah ke penurunan ketidak-murnian maksimum secara keseluruhan
    #Jika aturan stop-pemilahan telah dideklarasikan simpul t sebagai sebuah daun dan melabelinya dengan label kelas.
    # Jika tidak, bangkitkan dua simpul keturunan t_Y dan t_N dengan himpunan bagian yang bersesuaian dengan X_{tY} dan X_{tN}, tergantung pada jawaban untuk pertanyaan: apakah $x_{k0} \leq \alpha_{k0n0}$
- END

KETERANGAN

- Sebuah variasi dari ukuran ketidak-murnian simpul dapat ditentukan. Namun, seperti yang ditunjukkan dalam [Brei 841, sifat-sifat pohon terakhir yang dihasilkan tampaknya agak sensitif terhadap pilihan kriteria pemilahan. Namun demikian, ini lebih pada tugas tergantung masalah.
- Sebuah faktor penting dalam merancang pohon keputusan adalah ukurannya. Seperti halnya dengan perceptrons multi lapis, ukuran pohon harus cukup besar tapi tidak terlalu besar, jika ia cenderung untuk mempelajari rincian tertentu dari training himpunan dan menghasilkan kinerja generalisasi yang buruk. Pengalaman menunjukkan bahwa penggunaan nilai ambang batas untuk penurunan ketidak-murnian sebagai aturan stop-pemilahan tidak menyebabkan ukuran pohon yang tepat.

Beberapa kali pohon berhenti tumbuh, terlalu awal atau terlalu lambat. Pendekatan yang paling umum digunakan adalah menumbuhkan pohon sampai ukuran besar pertama dan kemudian memangkas simpul sesuai dengan kriteria pemangkasan. Hal ini mirip dalam filsafat dengan perceptrons pemangkasan multi lapis. Sejumlah pemangkasan kriteria telah diusulkan dalam literatur. Sebuah kriteria yang umum digunakan adalah untuk menggabungkan perkiraan probabilitas kesalahan dengan kompleksitas Istilah mengukur (misalnya, jumlah simpul terminal). Untuk lebih lanjut tentang masalah ini pembaca yang tertarik dapat merujuk ke [Brei 84, Rip1 94].

- Diskusi kita sejauh ini difokuskan pada jenis pohon OBCT. Partisi lebih umum dari ruang fitur, melalui hyperplanes tidak sejajar dengan sumbu, adalah mungkin melalui pertanyaan-pertanyaan dari jenis: apakah $\sum_{k=1}^{l} ckxk \leq \propto$?? Hal ini dapat mengakibatkan partisi ruang yang lebih baik. Namun, pelatihan sekarang menjadi lebih terlibat; lihat, misalnya, [Quin 93]..
- Konstruksi dari pohon keputusan fuzzy juga telah diusulkan, dengan membiarkan kemungkinan keanggotaan parsial dari vektor fitur di simpul yang membentuk struktur pohon. Fuzzifikasi dapat dicapai dengan menerapkan struktur fuzzy atas kerangka dasar dari sebuah pohon keputusan standar, lihat, misalnya, [Suar 99] dan referensi di dalamnya.

Contoh 4.1. Dalam tugas klasifikasi pohon, himpunan X_t berhubungan dengan simpul 1, berisi N_t = 10 vektor. Empat dari milik kelas ω_l, empat sampai kelas ω_2. dan dua untuk diklasifikasikan dalam tugas klasifikasi tiga kelas ω_3. Hasil pemilahan simpul menjadi dua himpunan bagian baru X_{tY}, dengan tiga vektor dari ω_l, dan satu dari ω_2 dan X_{tN} dengan satu vektor dari ω_1, tiga dari ω_2 dan dua dari ω_3. Tujuannya adalah untuk menghitung penurunan ketidak-murnian simpul setelah pemilahan. Kami memiliki

$$I(t) = -\frac{4}{10}\log_2\frac{4}{10} - \frac{4}{10}\log_2\frac{4}{10} - \frac{2}{10}\log_2\frac{2}{10} = 1.521$$

$$I(t_Y) = -\frac{3}{4}\log_2\frac{3}{4} - \frac{1}{4}\log_2\frac{1}{4} = 0.815$$

$$I(t_N) = -\frac{1}{6}\log_2\frac{1}{6} - \frac{3}{6}\log_2\frac{3}{6} - \frac{2}{6}\log_2\frac{2}{6} = 1.472$$

Oleh karena itu, penurunan ketidak-murnian setelah memisah

$$\Delta I(t) = 1.521 - \frac{4}{10}(0.815) - \frac{6}{10}(1.472) = 0.315$$

Untuk informasi lebih lanjut dan studi yang lebih dalam pemilah pohon keputusan pembaca yang berminat dapat berkonsultasi melalui buku seminal [Brei 84]. Contoh nonexhaustive dari kontribusi lanjutan di daerah tersebut adalah [Datt 85, Chou 91, Seth 90, Graj 86, Quin 93]. Sebuah pedoman komparatif terbaru untuk sejumlah teknik terkenal disediakan dalam [Espo 97].

Akhirnya, harus dinyatakan bahwa ada kesamaan yang erat antara keputusan pohon dan pengklasifikasi jaringan syaraf tiruan. Keduanya bertujuan membentuk batas keputusan yang kompleks dalam ruang fitur. Perbedaan utama terletak pada cara keputusan dibuat. Keputusan pohon menggunakan fungsi keputusan secara hirarkis terstruktur dalam mode sekuensial. Sebaliknya, jaringan syaraf memanfaatkan seperangkat lunak (belum final) keputusan secara paralel.

Selanjutnya, pelatihan dilakukan melalui filosofi yang berbeda. Namun, meskipun perbedaan mereka, telah menunjukkan bahwa pohon pengklasifikasi linear (dengan kriteria

pemecahan linear) dapat cukup dipetakan ke struktur multilayer perceptron [Seth 90, Seth 91, Taman 94].

Sejauh ini, dari sudut pandang kinerja, studi banding tampaknya memberikan keunggulan pada perceptrons multi lapis sehubungan dengan kesalahan klasifikasi, dan keunggulan pada pohon keputusan sehubungan dengan waktu pelatihan yang dibutuhkan [Brow 93].

4.19 DISKUSI

Bab saat ini adalah yang ketiga mengenai tahap proses klasifikasi. Meskipun kita belum menghabiskan daftar (sebagai sebuah kenyataan, kasus yang lebih sedikit akan dibahas pada bab-bab selanjutnya), kami merasa bahwa kami telah menyajikan kepada pembaca petunjuk yang paling populer saat ini digunakan untuk desain sebuah pemilah.

Kecenderungan lain yang menawarkan lebih banyak kemungkinan untuk desainer adalah untuk menggabungkan pemilah berbeda secara bersama-sama. Dengan demikian, kita dapat memanfaatkan kelebihan masing-masing dalam rangka mencapai kinerja yang lebih baik secara keseluruhan daripada yang dapat dicapai dengan menggunakan mereka masing-masing secara terpisah. Sebuah pengamatan penting yang membenarkan pendekatan semacam ini berikut ini. Dari (calon) pemilah yang berbeda yang kami desain untuk memilih salah satu yang sesuai dengan kebutuhan kita, salah satu hasil dalam performa terbaik, yaitu, klasifikasi tingkat kesalahan minimum. Namun, pemilah yang berbeda mungkin gagal (untuk mengklasifikasikan benar) pada pola yang berbeda. Artinya, bahkan pemilah "terbaik" bisa gagal pada pola-pola yang pemilah lain sukses di. Menggabungkan pemilah bertujuan mengeksploitasi- hal ini melengkapi informasi yang tampaknya berada di berbagai pengklasifikasi. Banyak masalah desain yang menarik sekarang datang ke tempat kejadian. Apa strategi seseorang harus mengadopsi untuk menggabungkan keluaran individu untuk mencapai kesimpulan final? Jika salah satu menggabungkan hasil sebagai berikut aturan produk, aturan penjumlahan, aturan min, max aturan atau aturan median? Haruskah semua pemilah diberi makan dengan vektor fitur yang sama atau harus vektor fitur yang berbeda dipilih untuk pengklasifikasi berbeda? Untuk melihat tentang teknik-teknik seperti pembaca dapat merujuk ke [Kitt 98, Pajak 00, Mill 99, Jain 00, Witt 88, Kunc 02] dan referensi di dalamnya.

Merujuk pada jenis spesifik strategi untuk menggabungkan pengklasifikasi. Di jantung metode meningkatkan kebohongan sehingga disebut pemilah "dasar". Ini adalah pemilah yang "lemah" dan itu sudah cukup untuk melakukan sedikit lebih baik daripada acak menebak. Suatu seri dari pengklasifikasi kemudian dirancang iteratif, mempekerjakan, setiap waktu, pemilah dasar, tetapi menggunakan himpunan bagian berbeda dari training himpunan, menurut sebuah distribusi iteratif dihitung (atau pembobotan atas sampel pelatihan diatur). Pada setiap iterasi, distribusi bobot dihitung memberikan penekanan pada sampel yang "paling sulit" (salah diklasifikasikan). Pemilah akhir diperoleh sebagai rata-rata tertimbang sesuai (suara) dari pengklasifikasi sebelumnya hierarkis dirancang. Ternyata, diberikan dalam jumlah yang memadai dan sampel iterasi pelatihan klasifikasi kesalahan kombinasi akhir dapat begitu kecil. Hal ini sangat mengesankan memang. Menggunakan pemilah sangat lemah sebagai dasar, seseorang dapat mencapai tingkat kesalahan yang relative kecil, dengan eksploitasi yang sesuai dari perilaku sampel pelatihan sehubungan dengan urutan pengklasifikasi dirancang [Scha 98]. Sebuah pengamatan yang lebih dalam pada pemilah ini mengungkapkan bahwa mereka pada dasarnya pemilah "margin". Yaitu, pada setiap langkah iterasi mereka mencoba untuk memaksimalkan margin sampel pelatihan dari permukaan keputusan, dan mereka akhirnya bertemu dengan distribusi margin di mana contoh yang paling memiliki margin besar, lihat, misalnya, [Maso 00, Scha 98]. Dari sudut pandang ini ada kesamaan dengan dukungan mesin vektor, yang juga mencoba memaksimalkan marjin

sampel pelatihan dari permukaan keputusan, lihat, misalnya, [Scha 98, Tikus 02].Memang benar bahwa sejumlah teknik yang tersedia adalah besar dan pengguna harus memilih apa yang lebih tepat untuk masalah di tangan. Tidak ada resep ajaib. Sebuah upaya penelitian besar telah difokuskan pada studi perbandingan berbagai pengklasifikasi dalam konteks aplikasi yang berbeda. Salah satu upaya yang paling luas adalah proyek Statlog [Mich 94], di mana berbagai pemilah diuji menggunakan sejumlah besar himpunan data yang berbeda. Selain itu, upaya penelitian telah dikhususkan untuk hubungan terurai dan kedekatan antara teknik berbeda. Banyak dari teknik ini berasal dari berbagai disiplin ilmu yang berbeda. Oleh karena itu, hingga beberapa tahun yang lalu, mereka dianggap independen. Baru-baru ini para peneliti telah mulai mengakui kesamaan mendasar antara pendekatan bervariasi. Bagi pembaca yang ingin menggali sedikit lebih dalam pertanyaan-pertanyaan ini, diskusi dan hasil disajikan dalam [Chen 94 Ripl 94, Ripl 96, Spec 90, Holm 97.Josh 97, Reyn 99] akan cukup memberikan pencerahan.

Singkatnya, hanya ujung yang dapat diberikan kepada desainer adalah bahwa semua teknik yang disajikan dalam buku ini masih merupakan pemain yang serius dalam game perancangan pemilah. Pilihan akhir tergantung pada tugas tertentu. *Bukti dari kue adalah saat memakan.*