5

Model Linear

> Sometimes an entirely inaccurate formula is a handy way to move you in the right direction if it offers simplicity.
>
> Scott Adams

Bab ini membahas tentang model linear (algoritma parametrik), sebagai contoh pembelajaran yang sederhana karena cukup mudah dipahami idenya. Bab ini juga membahas *error function* dan *stochastic gradient descent*.

5.1 Curve Fitting dan Error Function

Pertama, penulis ingin menceritakan tentang salah satu bentuk *utility function* untuk model matematis bernama *error function*. Fungsi ini sudah banyak diceritakan pada bab-bab sebelumnya secara deskriptif. Mulai bab ini, kamu akan mendapatkan pengertian lebih jelas secara matematis.

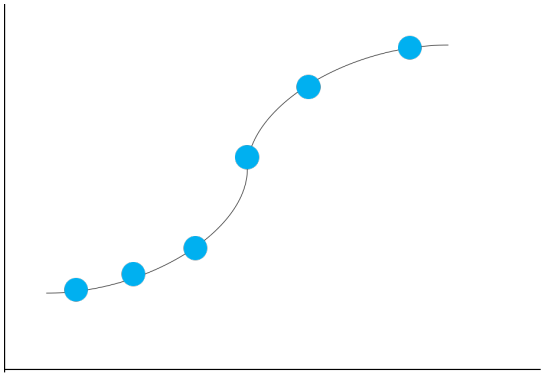
Error function paling mudah dijelaskan dalam permasalahan regresi. Masih ingat contoh bab sebelumnya tentang estimasi distribusi *Univariate Gaussian*? Ingat kembali konsep tersebut untuk mengerti bab ini. Diberikan (x, y) sebagai *random variable* berdimensi R^M dan R^N (keduanya berada pada Euclidean Space) yang merupakan parameter[1] bagi *probability density function* $q(x, y)$. Terdapat sebuah fungsi $f(x) \rightarrow y$, yang memetakan x ke y. Aproksimasi $f(x)$, sebut saja sebagai $g(x)$ adalah fungsi hasil regresi. Fungsi regresi g: $R^M \rightarrow R^N$ didefinisikan secara konseptual sebagai persamaan 5.1 [9].

$$g(x) = \int y \; q(y|x) dy \tag{5.1}$$

[1] *Which is subject to a simultaneous probability density function*

Persamaan 5.1 dibaca sebagai "*expectation of* y, *with the distribution of* q". Secara statistik, regresi dapat disebut sebagai ekspektasi untuk y berdasarkan/ dengan *input* x. Perlu diperhatikan, regresi adalah pendekatan sehingga belum tentu 100% benar (hal ini juga berlaku pada model *machine learning* pada umumnya).

Sebagai ilustrasi *curve fitting problem*, kamu diberikan fungsi $f(x)$ seperti pada Gambar. 5.1. sekarang fungsi $f(x)$ tersebut disembunyikan (tidak diketahui), diberikan contoh-contoh pasangan (x_i, y_i); $i = 1, 2, ..., 6$ adalah titik pada dua dimensi (titik sampel), seperti tanda bulat warna biru. Tugasmu adalah untuk mencari tahu $f(x)$!



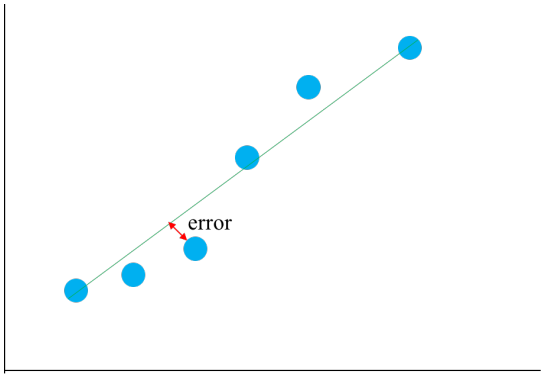
Gambar 5.1. Contoh fungsi Sigmoid.

Anggap dengan metode regresi, kamu berhasil melakukan pendekatan dan menghasilkan fungsi polinomial $g(x) = x \cdot \mathbf{w}$; seperti Gambar. 5.2 (garis berwarna hijau). Akan tetapi, fungsi approksimasi ini tidak 100% tepat sesuai dengan fungsi aslinya (ini perlu ditekankan). Jarak antara titik biru terhadap garis hijau disebut *error*.

Salah satu cara menghitung *error* fungsi $g(x)$ adalah menggunakan ***squared error function*** dengan bentuk konseptual pada persamaan 5.2. Estimasi terhadap persamaan tersebut disajikan dalam bentuk diskrit pada persamaan 5.3. (x_i, y_i) adalah pasangan *training data* (*input - desired output*). Nilai *squared error* dapat menjadi tolak ukur untuk membandingkan kinerja suatu *learning machine* (model). Secara umum, bila nilainya tinggi, maka kinerja dianggap relatif buruk; sebaliknya bila rendah, kinerja dianggap relatif baik. Hal ini sesuai dengan konsep *intelligent agent* [5].

$$E(g) = \int \int \|y - g(x)\|^2 \, q(x, y) \, dx \, dy \tag{5.2}$$

$$E(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} \|y_i - g(x_i, \mathbf{w})\|^2 \tag{5.3}$$



Gambar 5.2. Pendekatan fungsi Sigmoid.

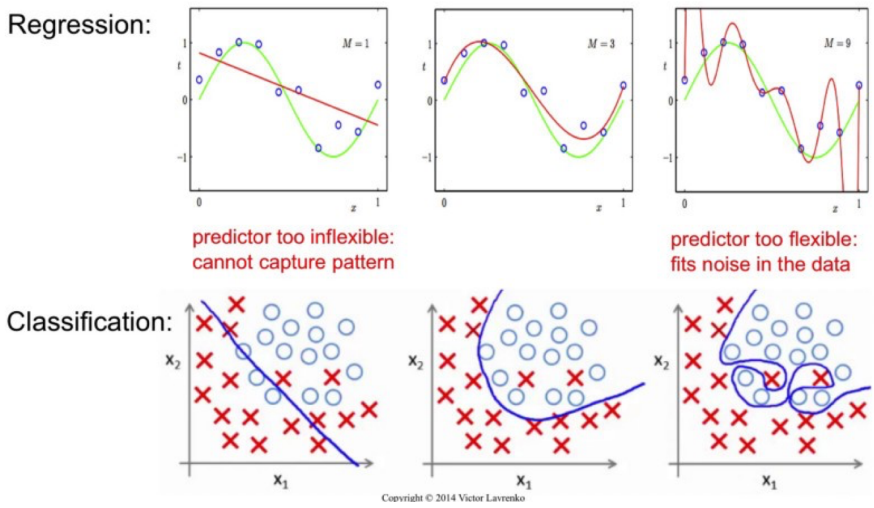
Ingat kembali bab 1, *learning machine* yang direpresentasikan dengan fungsi g bisa diatur kinerjanya dengan parameter *training* $\mathbf{w}$. *Square error* untuk *learning machine* dengan parameter *training* $\mathbf{w}$ diberikan oleh persamaan 5.3. (x_i, y_i) adalah pasangan *input-desired output*. Selain untuk menghitung ***square error*** pada *training data*, persamaan 5.3 juga dapat digunakan untuk menghitung *square error* pada testing data. Tujuan dari regresi/*machine learning* secara umum adalah untuk meminimalkan nilai *loss* atau *error* baik pada *training* maupun *unseen instances*. Secara umum, kita lebih ingin meminimalkan *loss*, dimana *error* dapat menjadi *proxy* untuk *loss*. Selain *error function*, ada banyak fungsi lainnya seperti *Hinge*, *Log Loss*, *Cross-entropy loss*, *Ranking loss* [1].

5.2 Overfitting dan Underfitting

Seperti yang sudah dijelaskan pada bab-bab sebelumnya (dan disinggung kembali pada subbab sebelumnya), tujuan *machine learning* adalah membuat model yang mampu memprediksi data yang belum pernah dilihat (*unseen instances*) dengan tepat; disebut sebagai generalisasi (*generalization*). Seperti yang sudah dijelaskan pada bab pertama, kita dapat membagi dataset menjadi *training*, *development*, dan *testing* dataset. Ketiga dataset ini berasal dari populasi yang sama dan dihasilkan oleh distribusi yang sama (*identically and independently distributed*). Dalam artian, ketiga jenis dataset mampu melambangkan (merepresentasikan) karakteristik yang sama[2]. Dengan demikian, kita ingin *loss* atau *error* pada *training*, *development*, dan *testing* bernilai kurang lebih bernilai sama (i.e., kinerja yang sama untuk data dengan karakteristik yang sama). Akan tetapi, ***underfitting*** dan ***overfitting*** mungkin terjadi.

[2] Baca teknik *sampling* pada buku statistika.

Underfitting adalah keadaan ketika kinerja model bernilai buruk baik pada *training* atau *development* maupun *testing data*. *Overfitting* adalah keadaan ketika kinerja model bernilai baik untuk *training* tetapi buruk pada *unseen data*. Hal ini diilustrasikan pada Gambar. 5.3. *Underfitting* terjadi akibat model yang terlalu tidak fleksibel (memiliki kemampuan yang rendah untuk mengestimasi variasi fungsi), sedangkan *overfitting* terjadi ketika model terlalu fleksibel (memiliki kemampuan yang terlalu tinggi untuk mengestimasi banyak fungsi) atau terlalu mencocokkan diri terhadap *training* data. Perhatikan kembali Gambar. 5.3, dataset asli diambil (*sampled*) dari fungsi polinomial orde-3. Model *underfitting* hanya mampu mengestimasi dalam orde-1 (kemampuan terlalu rendah), sedangkan model *overfitting* mampu mengestimasi sampai orde-9 (kemampuan terlalu tinggi).

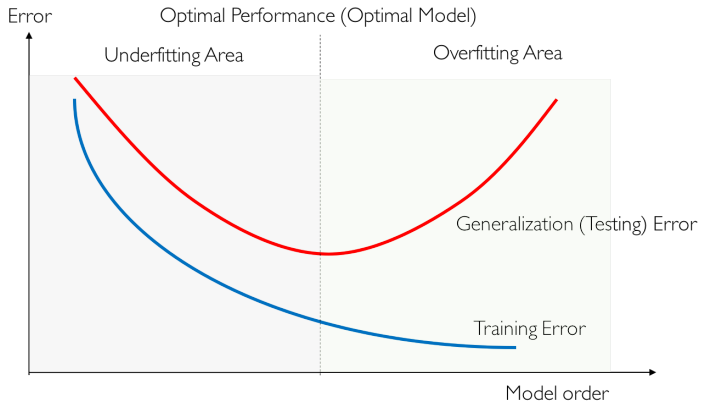


Gambar 5.3. *Underfitting* vs. *Overfitting*[3].

Apabila kita gambarkan grafik kinerja terhadap konfigurasi model (*model order*), fenomena *underfitting* dan *overfitting* dapat diilustrasikan seperti Gambar. 5.4. Model yang ideal adalah model yang memiliki kinerja yang baik pada *training*, *development*, dan *testing* data. Artinya, kita ingin perbedaan kinerja model pada berbagai dataset bernilai sekecil mungkin. Untuk menghindari *overfitting* atau *underfitting*, kita dapat menambahkan fungsi ***noise/bias*** (selanjutnya disebut *noise/bias* saja) dan regularisasi (subbab 5.8). Hal yang paling perlu pembaca pahami adalah untuk jangan merasa senang ketika model *machine learning* yang kamu buat memiliki kinerja baik pada *training data*. Kamu harus mengecek pada *development* dan *testing* data, serta

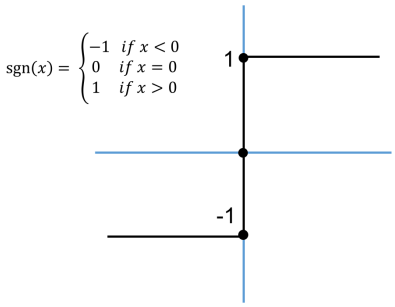
[3] `https://www.inf.ed.ac.uk/teaching/courses/iaml/slides/eval-2x2.pdf`

memastikan kesamaan karakteristik data (e.g., apakah *training* dan *testing* data benar diambil dari distribusi yang sama).



Gambar 5.4. *Selection Error.*

5.3 Binary Classification



Gambar 5.5. Fungsi *sign*.

Kamu sudah mendapat penjelasan pada bab-bab sebelumnya. *Binary classification* adalah mengklasifikasikan data menjadi dua kelas (*binary*). Contoh model linear sederhana untuk *binary classification* diberikan pada persamaan 5.4. Perhatikan, pada persamaan 5.4, suatu data direpresentasikan

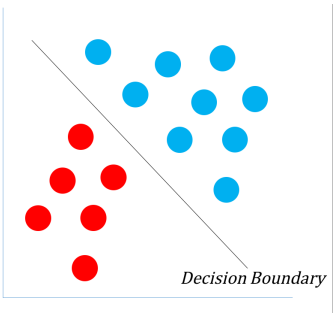
sebagai *feature vector* $\mathbf{x}$, dan terdapat *bias*[4] b. Klasifikasi dilakukan dengan melewatkan data pada fungsi yang memiliki parameter. Fungsi tersebut menghitung bobot setiap fitur pada vektor dengan mengalikannya dengan parameter (*dot product*). Persamaan 5.4 dapat ditulis kembali sebagai persamaan 5.5, dimana x_i merupakan elemen ke-i dari vektor $\mathbf{x}$. Fungsi ini memiliki *range* $[-\infty, \infty]$. Pada umumnya, kita menggunakan fungsi *sign* (sgn, Gambar. 5.5) untuk merubah nilai fungsi menjadi $[-1, 1]$ sebagai *output* (persamaan 5.6); dimana -1 merepresentasikan *input* dikategorikan ke kelas pertama dan nilai 1 merepresentasikan *input* dikategorikan ke kelas kedua.

$$f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{w} + b \tag{5.4}$$

$$f(\mathbf{x}) = x_1 w_1 + x_1 w_2 + \cdots + x_n w_n + b \tag{5.5}$$

$$\text{output} = \text{sgn}(f(\mathbf{x}) \tag{5.6}$$

Seperti halnya fungsi regresi, kita juga dapat menghitung performa *binary classifier* sederhana ini menggunakan *squared error function*, dimana nilai target fungsi berada pada range $[-1, 1]$. Secara sederhana, model *binary classifier* mencari ***decision boundary***, yaitu garis (secara lebih umum, *hyperplane*) pemisah antara kelas satu dan lainnya. Sebagai contoh, garis hitam pada Gambar. 5.6 adalah *decision boundary*.



Gambar 5.6. Contoh *decision boundary*.

5.4 Log-linear Binary Classification

Pada subbab sebelumnya, telah dijelaskan fungsi binary classifier memetakan data menjadi nilai $[-1, 1]$, dengan -1 merepresentasikan kelas pertama dan

[4] Berbeda dengan contoh pada subbab sebelumnya karena data adalah skalar. Penulis berharap pembaca dapat menginterpretasikan simbol berdasarkan konteks.

1 merepresentasikan kelas kedua. Tidak hanya kelas yang berkorespondensi, kita juga terkadang ingin tahu seberapa besar peluang data tergolong pada kelas tersebut. Salah satu alternatif adalah dengan menggunakan fungsi sigmoid dibanding fungsi *sign* untuk merubah nilai fungsi menjadi $[0, 1]$ yang merepresentasikan peluang data diklasifikasikan sebagai kelas tertentu (1 - nilai peluang, untuk kelas lainnya). Konsep ini dituangkan menjadi persamaan 5.7, dimana y merepresentasikan probabilitas, $\mathbf{x}$ merepresentasikan data (*feature vector*), dan b merepresentasikan *bias*. Ingat kembali materi bab 4, algoritma *Naive Bayes* melakukan hal serupa. Hasil fungsi sigmoid, apabila di-*plot* maka akan berbentuk seperti Gambar. 5.1 (berbentuk karakter "S").

$$y = \sigma(f(\mathbf{x})) = \frac{1}{1 + e^{-(\mathbf{x} \cdot \mathbf{w} + b)}} \tag{5.7}$$

5.5 Multi-class Classification

Subbab ini akan membahas tentang *multi-class classification*, dimana terdapat lebih dari dua kemungkinan kelas. Terdapat himpunan kelas C beranggotakan $\{c_1, c_2, \ldots, c_K\}$. Untuk suatu data dengan representasikan *feature vector*-nya, kita ingin mencari tahu kelas yang berkorespondesi untuk data tersebut. Contoh permasalahan ini adalah mengklasifikasi gambar untuk tiga kelas: *apel*, *jeruk*, atau *mangga*. Cara sederhana adalah memiliki tiga buah vektor parameter dan *bias* berbeda, $\mathbf{w}_{apel}$, $\mathbf{w}_{jeruk}$, $\mathbf{w}_{mangga}$, dan *bias* $b_{\{apel,jeruk,mangga\}}$. Untuk menentukan suatu data masuk ke kelas mana, kita dapat memprediksi skor tertinggi yang diberikan oleh operasi *feature vector* terhadap masing-masing vektor parameter. Konsep matematisnya diberikan pada persamaan 5.8, dimana $\hat{c}$ adalah kelas terpilih (keputusan), yaitu kelas yang memiliki nilai tertinggi.

$$\hat{c} = f(\mathbf{x}) = \arg\max_{c_i \in C} (\mathbf{x} \cdot \mathbf{w}_{c_i} + b_{c_i}) \tag{5.8}$$

Tiga set parameter $\mathbf{w}_{c_i}$ dapat disusun sedemikian rupa sebagai matriks $\mathbf{W} \in \mathbb{R}^{d \times 3}$, dimana d adalah dimensi *feature vector* ($\mathbf{x} \in \mathbb{R}^{1 \times d}$). Demikian pula kita dapat susun bias menjadi vektor $b \in \mathbb{R}^{1 \times 3}$ berdimensi tiga. Dengan demikian, persamaan 5.8 dapat ditulis kembali sebagai persamaan 5.9. $\hat{\mathbf{c}}$ adalah vektor yang memuat nilai fungsi terhadap seluruh kelas. Kita memprediksi kelas berdasarkan indeks elemen $\hat{\mathbf{c}}$ yang memiliki nilai terbesar (Persamaan 5.10).

$$\hat{\mathbf{c}} = f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{W} + \mathbf{b} \tag{5.9}$$

$$\text{kelas prediksi} = \arg\max_{\hat{c}_i \in \hat{\mathbf{c}}} \hat{c}_i \tag{5.10}$$

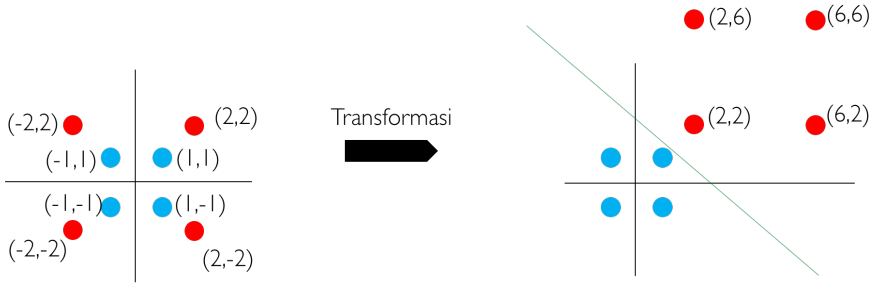
Seperti yang diceritakan pada subbab berikutnya, kita mungkin juga tertarik dengan probabilitas masing-masing kelas. Kita dapat menggunakan

fungsi *softmax*[5] untuk hal ini. Fungsi *softmax* mentransformasi $\hat{\mathbf{c}}$ agar jumlah semua nilainya berada pada range $[0, 1]$. Dengan itu, $\hat{\mathbf{c}}$ dapat diinterpretasikan sebagai distribusi probabilitas. Konsep ini dituangkan pada persamaan 5.11, dimana $\hat{c}_i$ adalah elemen vektor ke-i, melambangkan probabilitas masuk ke kelas ke-i.

$$\hat{c}_i = \frac{e^{(\mathbf{x}\cdot\mathbf{W}+\mathbf{b})_{[i]}}}{\sum_j e^{(\mathbf{x}\cdot\mathbf{W}+\mathbf{b})_{[j]}}} \tag{5.11}$$

5.6 Transformasi

Seperti yang sudah dijelaskan pada bab-bab sebelumnya. Alangkah baik apabila semua data bersifat *linearly separable*. Kenyataannya, kebanyakan data bersifat *non-linearly separable*. Dengan demikian, kita membutuhkan suatu fungsi untuk mentransformasi data, sebelum menggunakan model linear untuk mengklasifikasikan data. Sebagai contoh, perhatikan Gambar. 5.7. Pada gambar bagian kiri, terdapat empat titik yang *non-linearly separable*. Titik-titik itu ditransformasi sehingga menjadi gambar bagian kanan. Fungsi transformasi yang digunakan diberikan pada persamaan 5.12.



Gambar 5.7. Contoh transformasi [7].

$$\phi(x, y) = \begin{cases} \sqrt{x^2 + y^2} \geq 2 \rightarrow (4 - x + \|x - y\|, 4 - x + \|x - y\|) \\ \sqrt{x^2 + y^2} \leq 2 \rightarrow (x, y) \end{cases} \tag{5.12}$$

Secara umum, fungsi transformasi tidaklah sesederhana contoh yang diberikan. Fungsi transformasi pada umumnya menambah dimensi data (misal dari dua dimensi, menjadi tiga dimensi). Beberapa fungsi transformasi (dikenal juga dengan istilah *kernel*) yang terkenal diantaranya[6]:

[5] `https://en.wikipedia.org/wiki/Softmax_function`
[6] `https://en.wikipedia.org/wiki/Kernel_method`

1. Fisher Kernel
2. Graph Kernel
3. Kernel Smoother
4. Polynomial Kernel
5. Radial Basis Function Kernel
6. String Kernel

Untuk penjelasan masing-masing *kernel*, silahkan membaca literatur lain lebih lanjut. Model linear yang memanfaatkan fungsi-fungsi *kernel* ini adalah *support vector machine* (SVM) [25]. Perhatikan, algorithma SVM sebenarnya sangatlah penting. Akan tetapi, perlu kami informasikan bahwa buku ini tidak memuat materi SVM karena penulis belum mengerti sampai seluk beluk SVM[7]. Dengan demikian, kami harap pembaca dapat mencari referensi lain tentang SVM.

5.7 Pembelajaran sebagai Permasalahan Optimisasi

Salah satu tujuan dari pembelajaran (*training*) adalah untuk meminimalkan *error* sehingga kinerja *learning machine* (model) diukur oleh *square error*. Dengan kata lain, *utility function* adalah meminimalkan *square error*. Secara lebih umum, kita ingin meminimalkan *loss* (*squared error function* adalah salah satu *loss function*), diilustrasikan pada persamaan 5.13, dimana θ adalah *learning parameter*[8], dan $\mathcal{L}$ adalah *loss function*. Perubahan parameter dapat menyebabkan perubahan *loss*. Karena itu, *loss function* memiliki θ sebagai parameternya.

$$\hat{\theta} = \arg\min_{\theta} \mathcal{L}(\theta) \tag{5.13}$$

dimana $\hat{\theta}$ adalah nilai parameter paling optimal. Perhatikan, "arg min" dapat juga diganti dengan "arg max" tergantung **optimisasi** apa yang ingin dilakukan.

Sekarang, mari kita hubungkan dengan contoh yang sudah diberikan pada subbab sebelumnya. Kita coba melakukan estimasi *minimum square error*, dengan mencari nilai *learning parameters* $\mathbf{w}$ yang meminimalkan nilai *error* pada model linear (persamaan 5.14)[9]. Terdapat beberapa cara untuk memi-

[7] Penulis memutuskan tidak membahas SVM, dibanding pembahasan pada level *superficial*.

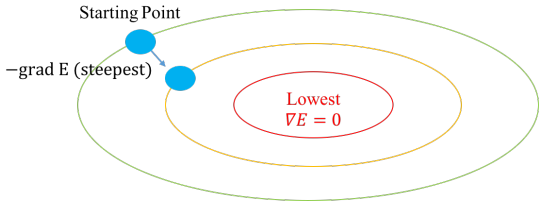
[8] Umumnya dapat berupa skalar, vektor, atau matriks. Demi istilah yang lebih generik, kita gunakan θ.

[9] $\mathbf{w}$ boleh diganti dengan $\mathbf{W}$, saat ini penulis menggunakan vektor untuk menyederhanakan pembahasan.

malkan *square error*. Yang penulis akan bahas adalah *gradient-based method*. ***Stochastic gradient descent*** akan dibahas untuk meminimalkan *error*[10].

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} E(\mathbf{w}) \tag{5.14}$$

Bayangkan kamu sedang berada di puncak pegunungan. Kamu ingin mencari titik terendah pegunungan tersebut. Kamu tidak dapat melihat keseluruhan pegunungan, jadi yang kamu lakukan adalah mencari titik terendah sejauh mata memandang, kemudian menuju titik tersebut dan menganggapnya sebagai titik terendah. Layaknya asumsi sebelumnya, kamu juga turun menuju titik terendah dengan cara melalui jalanan dengan kemiringan paling tajam, dengan anggapan bisa lebih cepat menuju ke titik terendah [9]. Sebagai ilustrasi, perhatikan Gambar. 5.8!



Gambar 5.8. *Stochastic Gradient Descent.*

Jalanan dengan kemiringan paling tajam adalah $-grad\ E(\mathbf{w})$, dimana $E(\mathbf{w})$ adalah nilai *error* saat model memiliki parameter $\mathbf{w}$. Dengan definisi $grad\ E(\mathbf{w})$ diberikan pada persamaan 5.15 dan persamaan 5.16, dimana w_i adalah nilai elemen vektor ke-i.

$$grad\ E(\mathbf{w}) = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, ..., \frac{\partial E}{\partial w_3} \right) \tag{5.15}$$

$$\frac{\mathrm{d}\mathbf{w}}{\mathrm{d}t} = -grad\ E(\mathbf{w}); t = time \tag{5.16}$$

Ingat kembali materi diferensial. Gradien adalah turunan (diferensial) fungsi. Untuk mencari turunan paling terjal, sama halnya mencari nilai −gradient terbesar. Dengan demikian, menghitung $-grad\ E(\mathbf{w})$ terbesar sama dengan jalanan turun paling terjal.

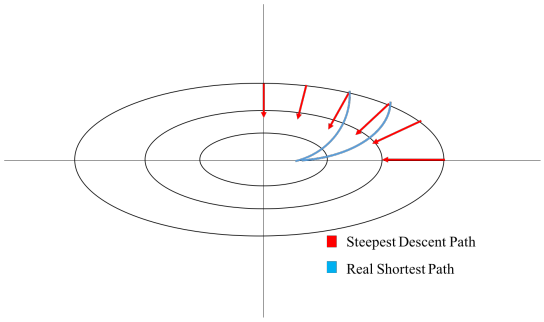
Tentunya seiring berjalannya waktu, kita mengubah-ubah parameter $\mathbf{w}$ agar kinerja model optimal. Nilai optimal diberikan oleh turunan $\mathbf{w}$ terhadap waktu, yang bernilai sama dengan $-grad\ E(\mathbf{w})$. Bentuk diskrit persamaan 5.16 diberikan pada persamaan 5.17

[10] Konsep *Hill Climbing* dapat digunakan untuk memaksimalkan *utility function*. Konsep tersebut sangat mirip dengan *gradient descent* `https://en.wikipedia.org/wiki/Hill_climbing`.

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \; grad \; E(\mathbf{w}(t)) \tag{5.17}$$

dimana η disebut ***learning rate***, dan $\mathbf{w}(t)$ adalah nilai $\mathbf{w}$ saat waktu/iterasi t. *Learning rate* digunakan untuk mengatur seberapa pengaruh keterjalan terhadap pembelajaran. Silahkan mencari sumber tambahan lagi agar dapat mengerti *learning rate* secara lebih dalam/matematis. Pada implementasi, η juga sering diubah-ubah nilainya sepanjang waktu. Semakin kita sudah dekat dengan tujuan (titik *loss* terendah), kita mengurangi nilai η (ibaratnya seperti mengerem) [26].

Walaupun kamu berharap bisa menuju titik terendah dengan menelusuri jalan terdekat dengan kemiringan paling tajam, tapi kenyataanya hal tersebut bisa jadi bukanlah jalan tercepat, seperti yang diilustrasikan pada Gambar. 5.9. Warna merah melambangkan jalan yang dilalui *gradient descent*, sementara warna biru melambangkan jalanan terbaik (tercepat).

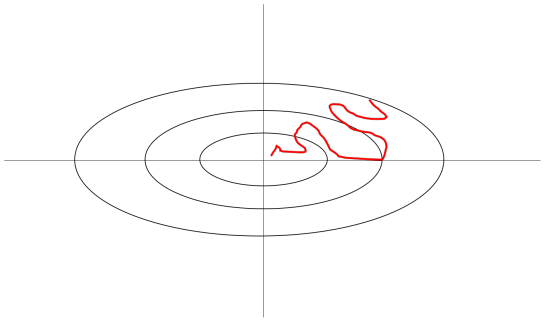


Gambar 5.9. *Stochastic Gradient Descent* 2.

Pandangan kita yang terbatas layaknya kita tidak bisa melihat keseluruhan pengunungan secara keseluruhan, kita juga tidak bisa melihat keseluruhan nilai *error* untuk semua parameter $\mathbf{w}$. Secara filosofis, hal tersebut juga berlaku saat membaca buku, oleh karena itu sebaiknya kamu membaca beberapa buku saat belajar.

Dalam *local point of view*, *steepest gradient descent* adalah cara tercepat menuju titik terendah, tetapi tidak pada *global point of view*. Kita dapat macet/berhenti saat sudah mencapai *local minima*, yaitu nilai minimum pada suatu daerah lokal saja. Untuk menghindari hal tersebut, kita menggunakan *learning rate* (η). Apabila nilai *learning rate* (η) pada persamaan 5.17 relatif kecil, maka dinamika perubahan parameter $\mathbf{w}$ juga kecil. Tetapi, bila nilainya besar, maka jalanan menuju titik terendah akan bergoyang-goyang (*swing*), seperti pada Gambar. 5.10.

Untuk mengontrol *learning parameter* $\mathbf{w}$ sehingga memberikan nilai $E(\mathbf{w})$ terendah, persamaan *steepest gradient descent* dapat ditambahkan dengan momentum (α) pada persamaan 5.18. Alfa adalah momentum karena dika-



Gambar 5.10. *Swing.*

likan dengan hasil perbedaan descent pada tahap sebelumnya. Alfa adalah parameter kontrol tambahan untuk mengendalikan *swing* yang sudah dibahas sebelumnya.

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \; grad \; E(\mathbf{w}(t)) + \alpha(\Delta \mathbf{w}) \tag{5.18}$$

Apabila gradien bernilai 0, artinya model sudah berada pada titik *local/global optimum*. Kondisi ini disebut sebagai **konvergen** (*converging*). Sementara model yang tidak menuju titik optimal, malah menuju ke kondisi yang semakin tidak optimum, disebut **divergen** (*diverging*).

5.8 Regularization

Gradient-based method mengubah-ubah parameter model **w** sehingga *loss/error* dapat diminimalisir. Perhatikan kembali Gambar. 5.2, ibaratnya agar fungsi aproksimasi kita menjadi sama persis dengan fungsi asli pada Gambar. 5.1. Perhatikan, karena nilai **w** berubah-ubah seiring waktu, bisa jadi urutan *training data* menjadi penting[11]. Pada umumnya, kita menggunakan *batch method*. Hal ini akan kamu lebih mengerti setelah mambaca bab 9.

Pada kenyataan, model yang kita hasilkan bisa jadi *overfitting*. Yaitu memiliki kinerja baik pada *training data*, tetapi memiliki kinerja buruk untuk *unseen data*. Salah satu cara menghindari *overfitting* adalah dengan menggunakan *regularization*. Idenya adalah untuk mengontrol kompleksitas parameter (i.e. konfigurasi parameter yang lebih sederhana lebih baik) . Dengan ini, objektif *training* pada persamaan 5.13 dapat kita ubah menjadi persamaan 5.19, dimana $\mathrm{R}(\theta)$ adalah fungsi *regularization* dan λ adalah parameter kontrol.

$$\theta = \arg\min_{\theta} \mathcal{L}(\theta) + \lambda \mathrm{R}(\theta) \tag{5.19}$$

[11] Baca buku Yoav Goldberg [1] untuk mendapat penjelasan lebih baik.

Pilihan umum untuk fungsi *regularization* pada umumnya adalah L_2 dan L_1 norm. L_2 *regularization* menggunakan jumlah kuadrat dari *Euclidean norm*[12] seluruh parameter, seperti pada persamaan 5.20. Sedangkan, L_1 *regularization* menggunakan jumlah dari nilai *absolute-value norm* seluruh parameter, diberikan pada persamaan 5.21.

$$\mathrm{R}(\theta) = \|\mathbf{w}\|_2^2 = \sum_i (w_i)^2 \tag{5.20}$$

$$\mathrm{R}(\theta) = \|\mathbf{w}\| = \sum_i (w_i) \tag{5.21}$$

Izinkan saya mengutip pernyataan dari buku Yoav Goldberg [1] halaman 31 secara langsung menggunakan bahasa Inggris agar tidak ada makna yang hilang.

Convexity. In gradient-based optimization, it is common to distinguish between *convex* (or *concave*) functions and *non-concave* functions. A *convex function* is a function whose second-derivative is always non-negative. As a consequence, convex functions have a single minimum point. Similarly, *concave functions* are functions whose second-derivatives are always negative or zero, and as a consequence have a single maximum point. Convex (concave) functions have the property that they are easy to minimize (maximize) using gradient-based optimization–simply follow the gradient until an extremum point is reached, and once it is reached we know we obtained the global extremum point. In contrast, for functions that are neither convex or concave, a gradient-based optimization procedure may converge to a local extremum point, missing the global optimum.

5.9 Bacaan Lanjutan

Kami harap pembaca mampu mengeksplorasi materi *kernel method* dan *support vector machine* (SVM). Kami tidak mencantumkan kedua materi tersebut karena belum mampu menulis penurunan matematisnya secara detil dengan baik. Kami sarankan kamu membaca pranala `https://www.svm-tutorial.com/` karena ditulis dengan cukup baik. Mengerti materi SVM dan *convex optimization* secara lebih dalam akan sangat membantu pada bab-bab berikutnya. Selain itu, kami juga menyarankan pembaca untuk melihat kedua pranala tambahan tentang *learning rate* dan momentum:

1. `http://users.ics.aalto.fi/jhollmen/dippa/node22.html`
2. `http://www.willamette.edu/~gorr/classes/cs449/momrate.html`

[12] `https://en.wikipedia.org/wiki/Norm_(mathematics)`

Soal Latihan

5.1. Hill Climbing
Baca dan jelaskanlah konsep Hill Climbing!

5.2. Variasi Optimisasi
Baca dan jelaskanlah variasi konsep optimisasi lain, selain *stochastic gradient descent*!

5.3. Convex Optimization
Bacalah literatur yang memuat materi tentang *convex optimization*! Jelaskan pada teman-temanmu apa itu fungsi *convex* dan *concave*, tidak lupa isi materi yang kamu baca!