CSCM603130: Sistem Cerdas
Artificial Neural Networks

Fariz Darari, Aruni Yasmin Azizah

Fakultas Ilmu Komputer
Universitas Indonesia

2019/2020 • Semester Ganjil

Referensi:

Artificial Intelligence A Modern Approach, S. Russell & P. Norvig, Prentice Hall, 2010.

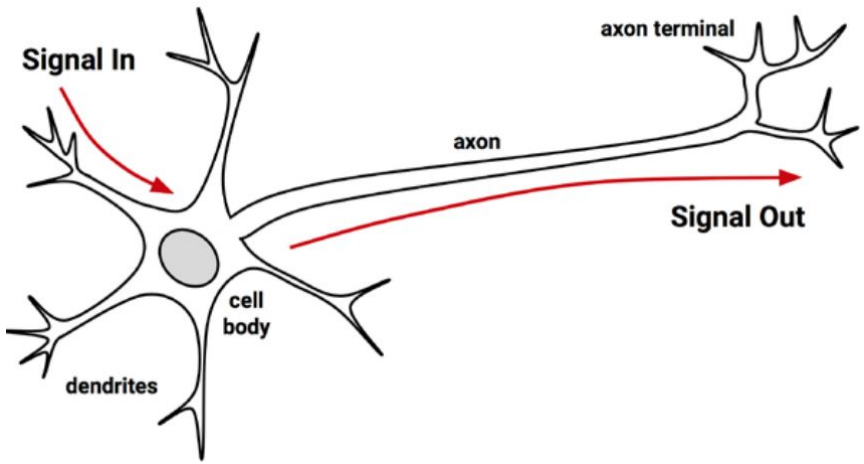
Machine Learning, T. Mitchell, McGraw Hill, 1997.

Outline

Outline

Motivasi: Connectionism

Neuron pada otak manusia:

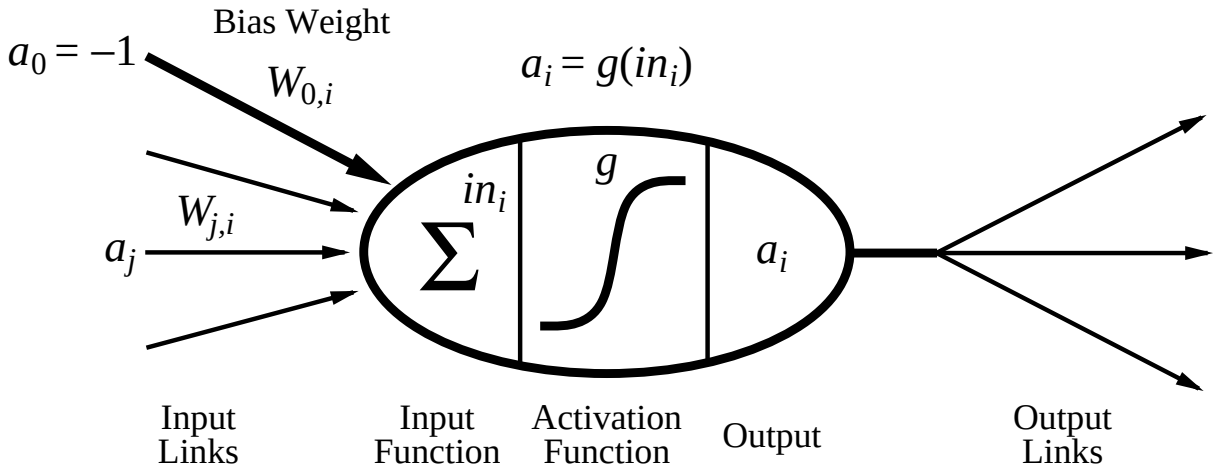


Sumber: Machine Learning with R, B. Lantz, 2015.

- Sinyal elektrik yang masuk, berupa ion, diterima dendrit.
- Badan sel saraf mengakumulasi sinyal masukan. Ketika mencapai threshold, sel menembakkan ion menjadi sinyal elektrik luaran yang dikirim melalui axon.
- Pada terminal axon, sinyal disampaikan lagi ke sel saraf lainnya melalui sinaps.

Motivasi: Connectionism

Model matematis sederhana dari neuron.



- Input diterima neuron.
- Input diakumulasi dengan input function. Melalui fungsi aktivasi, dihasilkan output a_i.
- Fungsi aktivasi g bisa berupa fungsi sigmoid, fungsi unit step, fungsi linier (identitas), fungsi saturated linier, dst.
- Neural network merupakan kumpulan unit atau node (dari unit input hingga unit output) yang terhubung (connected) dan membentuk topologi neuron.

Neural Network

Properti dari Artificial Neural Network (ANN):

- Threshold switching unit yang bekerja seperti dalam sel saraf.
- Unit yang terkoneksi memiliki bobot (weight).
- Proses yang memiliki tingkat distribusi dan paralel yang tinggi.
- Menitikberatkan pada penyesuaian bobot secara otomatis.

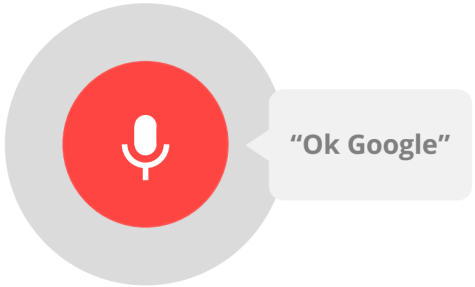
Kapan menggunakan Neural Network?

- input bernilai diskrit maupun real yang bersifat *high-dimensional*
- output bernilai diskrit maupun real
- output dapat dihasilkan dalam vektor
- input memiliki noise
- bentuk dari fungsi target ($f(x)$) tidak diketahui
- proses untuk mendapatkan hasil (atau *readability*) tidak penting (*black box*).

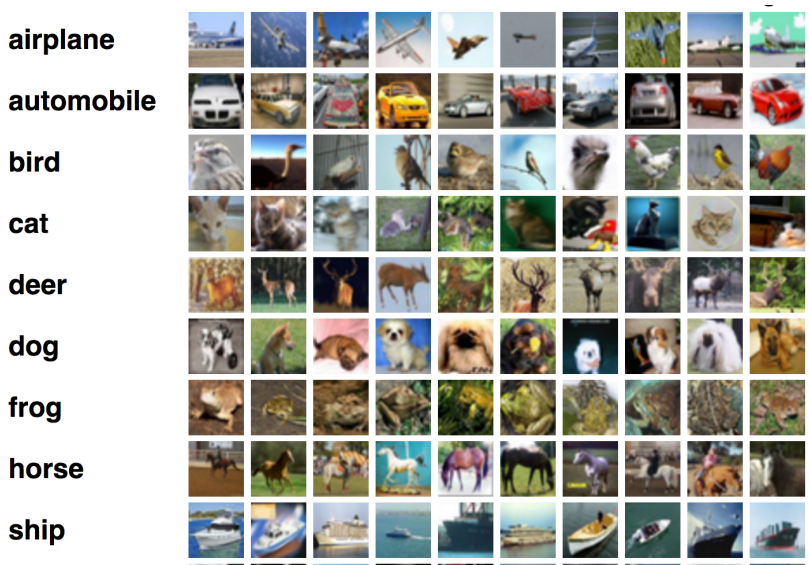
Neural Network

Contoh:

- Speech recognition



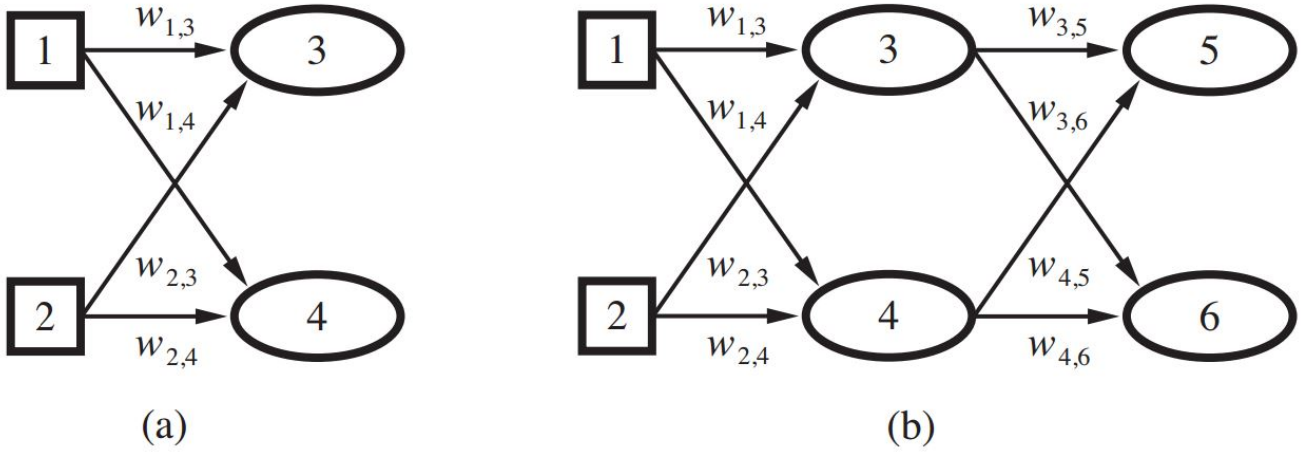
- Klasifikasi gambar



Outline

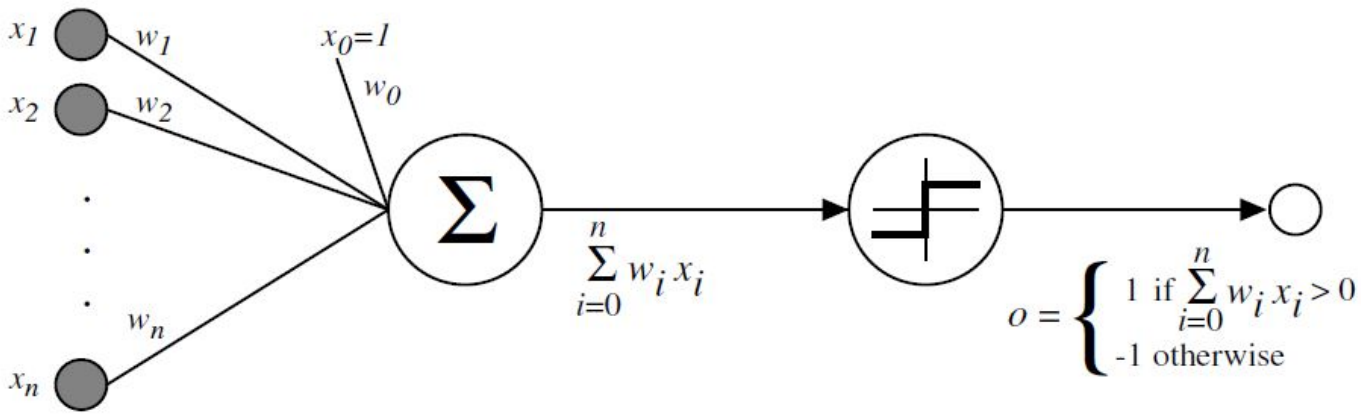
Perceptron

Network dengan semua input terkoneksi ke output disebut single-layer neural network, atau perceptron network.



- Network (a) merupakan perceptron network dengan 2 unit input dan 2 unit output.
- Network (b) merupakan neural network (multi-layer perceptrons), dengan 2 unit input dan memiliki 1 hidden layer yang terdiri atas 2 unit.

Perceptron



Contoh fungsi aktivasi:

$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \ldots + w_n x_n > 0 \\ -1 & \text{otherwise .} \end{cases}$$

Atau, notasinya dapat kita sederhanakan menjadi:

$$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w}.\vec{x} > 0 \\ -1 & \text{otherwise .} \end{cases}$$

Perceptron dengan Dua Input

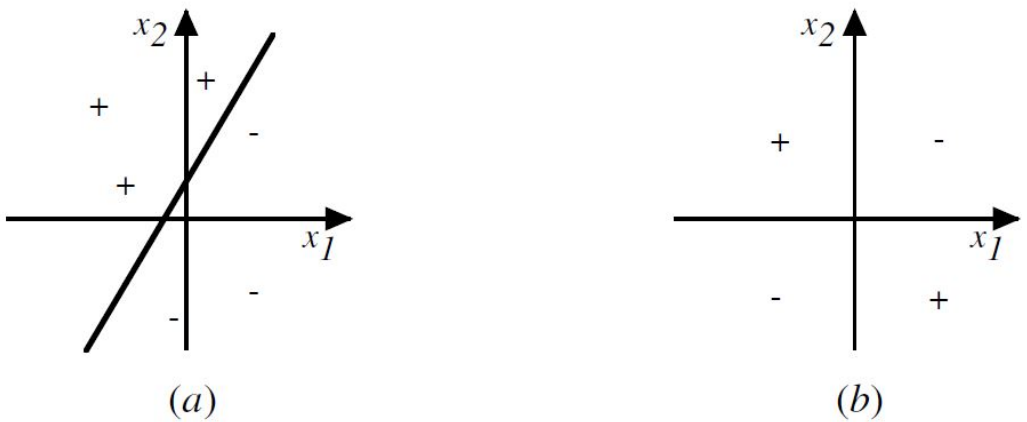
Rancanglah sebuah perceptron yang merepresentasikan fungsi:

- $g(x_1, x_2) = AND(x_1, x_2)$
- $g(x_1, x_2) = OR(x_1, x_2)$
- $g(x_1, x_2) = XOR(x_1, x_2)$

Perceptron dengan Dua Input

Rancanglah sebuah perceptron yang merepresentasikan fungsi:

- $g(x_1, x_2) = AND(x_1, x_2)$
- $g(x_1, x_2) = OR(x_1, x_2)$
- $g(x_1, x_2) = XOR(x_1, x_2)$



(a) sebuah perceptron yang mengklasifikasikan training examples dengan benar.
(b) training examples yang tidak *linearly separable* dan tidak dapat diklasifikasi dengan benar.

Outline

Perceptron Training Rule

- Sebelum memulai learning, tentukan nilai *weight* secara acak, nilai *bias* (umumnya bernilai 1), *learning rate* berupa bilangan real yang sangat kecil (mis. 0.1), dan *activation function* yang digunakan.
- Secara iteratif, terapkan perceptron pada setiap sample, dan update nilai *weight* ketika salah men-klasifikasi suatu sample.

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = \eta(t - o)x_i$$

Dimana:

  - t merupakan nilai target $c(\vec{x})$
  - o merupakan output perceptron $o(\vec{x})$
  - η merupakan *learning rate*
  - x_i adalah nilai input untuk fitur ke-i dan bobotnya w_i, dengan $i \neq 0$,
  - x_0 adalah nilai bias, dengan bobotnya w_0,

- Iterasi dijalankan sampai seluruh sample terklasifikasi dengan benar.

Perceptron Training Rule

- Perceptron training rule dapat dibuktikan akan konvergen menghasilkan vektor *weight* yang mengklasifikasikan semua training samples dengan benar, jika training samples bersifat linearly separable.
  - Jika $o = t$, perceptron sudah dapat meng-klasifikasi dengan benar, maka tidak ada *weight* yang perlu diupdate.
  - Jika $o < t$, misal $o = -1$ dan $t = +1$, w_i dinaikkan agar o mendekati nilai t.
  - Jika $o > t$, misal $o = +1$ dan $t = -1$, *weight* diturunkan agar o mendekati nilai t.
  - Jika η terlalu besar, kenaikan/penurunan o akan terlalu jauh sehingga sulit mendekati nilai t. Maka η yang cukup kecil membantu perceptron mencapai o yang mendekati nilai t.
- Namun, konvergen tidak dapat dipastikan jika:
  - training samplesnya tidak linearly separable

Outline

Delta Rule

Keterbatasan perceptron: gagal dalam menemukan *weight* yang tepat jika datanya non-linearly separable. Solusi → Delta Rule.

Ide Dasar Delta Rule

mencari vektor *weight* dalam *hypothesis space*, hingga mendapatkan hypothesis yang paling fit dengan training samples menggunakan gradient descent.

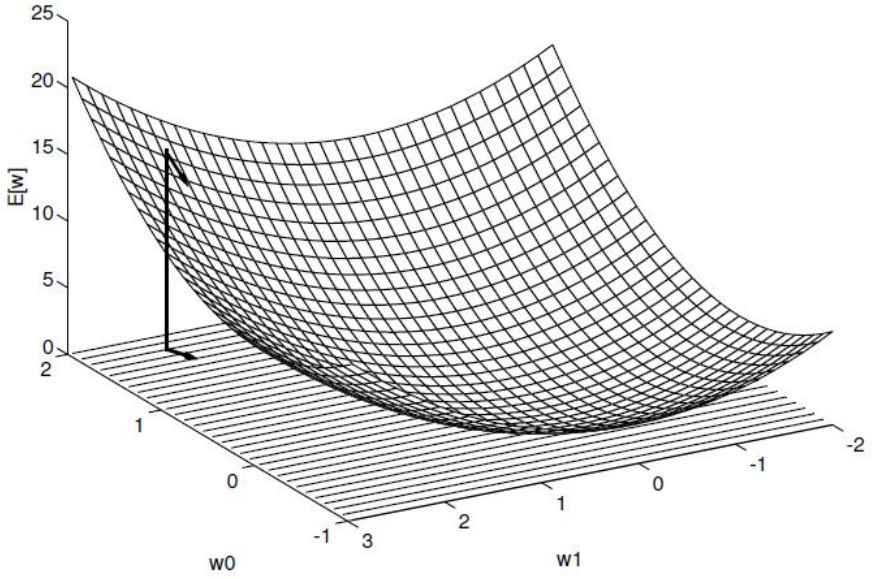
- Sebuah *unthresholded perceptron* (linear unit) menghasilkan output berupa kombinasi linier:

$$\begin{aligned} o(\vec{x}) &= \vec{w}.\vec{x} \\ &= w_0 + w_1.x_1 + w_2.x_2 + \ldots + w_n.x_n \end{aligned}$$

- Salah satu cara menghitung *training error* dari sebuah hipotesis (atau error dari vektor *weight*): setengah dari jumlah semua selisih antara target dan output yang telah dikuadratkan, untuk setiap sample d dari training samples D.

$$E(\vec{w}) \equiv \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$$

Gradient Descent



- Mencari $\vec{w}$ pada hypothesis space untuk meminimalkan error. (Garis vertikal menggambarkan error dari penggunaan w_0 dan w_1, relatif terhadap training samples)
- *Gradient* dari E terhadap vektor $\vec{w}$: turunan E terhadap setiap komponen pada vektor $\vec{w}$

$$\nabla E(\vec{w}) \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n}\right]$$

- $\nabla E(\vec{w})$: *steepest increase*, $-\nabla E(\vec{w})$: *steepest decrease*.

$$\vec{w} \leftarrow \vec{w} + \Delta\vec{w} \qquad w_i \leftarrow w_i + \Delta w_i$$

$$\Implies$$

$$\Delta\vec{w} = -\eta\nabla E(\vec{w}) \qquad \Delta w_i = -\eta\frac{\partial E}{\partial w_i}$$

Gradient Descent

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_d (t_d - o_d)^2 \\
&= \frac{1}{2}\sum_d \frac{\partial}{\partial w_i}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_d 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d) \\
&= \sum_d (t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}.\vec{x_d}) \\
\frac{\partial E}{\partial w_i} &= \sum_d (t_d - o_d)(-x_{i,d})
\end{aligned}$$

$x_{i,d}$ merupakan nilai input (fitur) ke-i untuk training sample ke-d.

Error Gradient untuk Linear Unit

$$\frac{\partial E}{\partial w_i} = -\sum_{d \in D}(t_d - o_d)x_{i,d}$$

Weight update rule dengan Gradient Descent

$$\Delta w_i = -\eta\frac{\partial E}{\partial w_i} = \eta(t_d - o_d)x_{i,d}$$

Gradient Descent

Algoritma Gradient Descent

GRADIENT-DESCENT(*training_examples*, η)

- Inisialisasi setiap nilai w_i dengan bilangan acak yang bernilai kecil.
- Selama kondisi terminasi belum tercapai, lakukan:
  - inisialisasi setiap Δw_i dengan 0.
  - untuk setiap $\langle \vec{x}, t \rangle$ dari training samples, lakukan:
    - masukkan $\vec{x}$ ke dalam unit, dan hitung nilai output
    - untuk setiap unit linear *weight* w_i, hitung:

$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$$

  - untuk setiap unit linear *weight* w_i, hitung:

$$w_i \leftarrow w_i + \Delta w_i$$

Tiap training example merupakan pasangan dari $\langle \vec{x}, t \rangle$, dimana $\vec{x}$ merupakan vektor input dan t adalah nilai target output. η merupakan learning rate

Gradient Descent

- Karena permukaan error (terlihat pada visualisasi hypothesis space) memiliki hanya 1 global minimum, maka algoritma Gradient Descent akan konvergen ke 1 vektor weight dengan error minimum, dengan diberikan η yang cukup kecil, walaupun training sample tidak linearly separable.
- Jika η terlalu besar, pencarian hypothesis dengan gradient descent berisiko overstep global minimum. Maka, modifikasi yang biasa dilakukan adalah menurunkan η secara bertahap ketika jumlah step meningkat.

Single-layer Networks – Summary

Dua algoritma yang secara iteratif menghitung *weight* perceptron.

- Perceptron training rule
  - Update *weight*: berdasarkan error pada fungsi *thresholded perceptron*
  - *Weight* konvergen menuju hipotesis yang sempurna membagi training examples
  - Konvergensi dapat dijamin jika training examples bersifat linearly separable dan η yang digunakan cukup kecil
- Delta rule dan gradient descent pada data yang tidak linearly separable
  - Update *weight*: berdasarkan error pada kombinasi linier dari input dalam fungsi *unthresholded perceptron*
  - *Weight* konvergen menuju hipotesis dengan error terkecil
  - Tetapi konvergensi dapat dicapai dengan η yang digunakan cukup kecil walaupun training examplesnya tidak linearly separable

Local Minima pada Gradient Descent

- Jika terdapat beberapa local minimum, tidak ada jaminan bagi gradient descent (batch) untuk mendapatkan global minimum.
- Untuk mengatasinya, diperkenalkan incremental gradient descent.

Ide dasar Incremental Gradient Descent

Mengaproksimasi gradient descent dengan *updating weight* secara *incremental* berdasarkan perhitungan error pada setiap training sample.

Incremental Gradient Descent

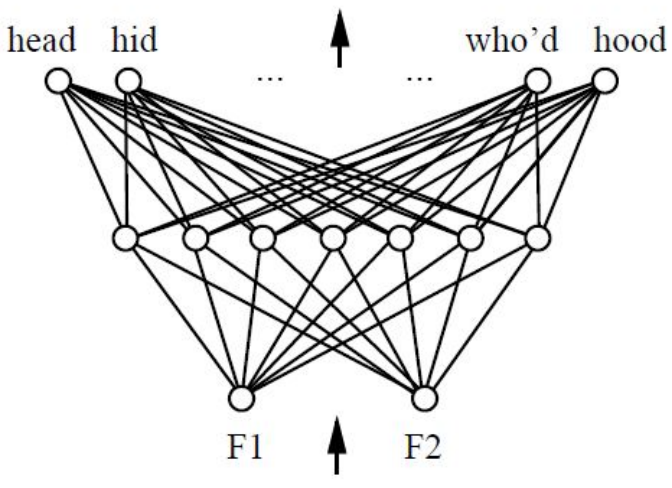
Batch mode gradient descent : Lakukan selama kondisi terminasi belum tercapai 1. Hitung gradient $\nabla E_D[\vec{w}]$ 2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_D[\vec{w}]$	Diketahui $E_D[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$
Incremental mode gradient descent : Lakukan selama kondisi terminasi belum tercapai • Untuk setiap training example$d \in D$ 1. Hitung gradient $\nabla E_d[\vec{w}]$ 2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_d[\vec{w}]$	Diketahui $E_d[\vec{w}] \equiv \frac{1}{2} (t_d - o_d)^2$

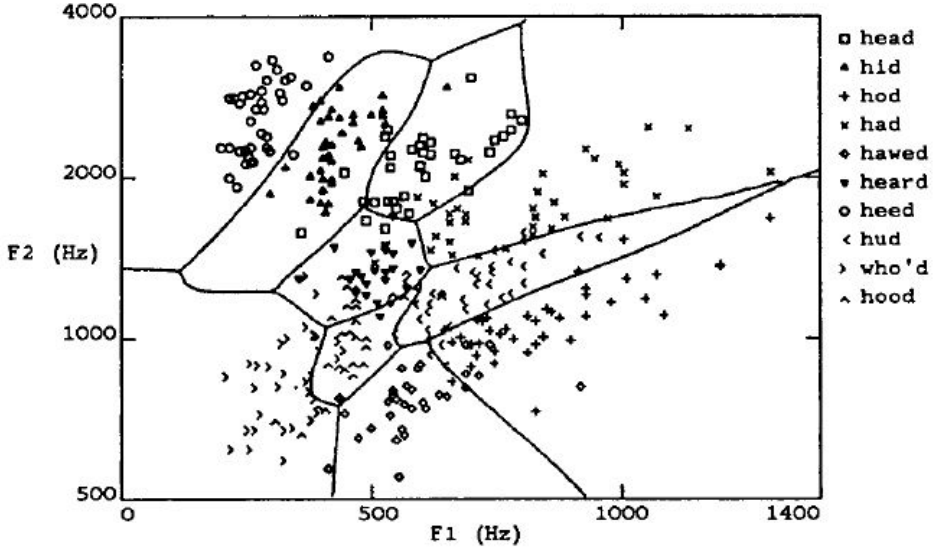
- *Incremental gradient descent* dapat mengaproksimasi *Batch gradient descent* cukup dekat jika η dibuat cukup kecil.
- Karena *incremental gradient descent* menggunakan variasi $\nabla E_d(\vec{w})$ daripada $\nabla E(\vec{w})$, pencarian hipotesis dapat terhindar dari local minimum.

Outline

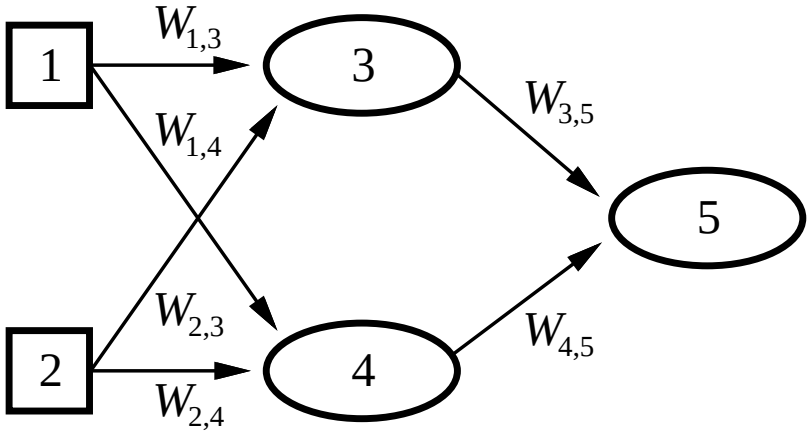
Multilayer Networks

Contoh multilayer networks untuk speech recognition dalam membedakan 10 pengucapan pada konteks "h_d"





Contoh Pengolahan pada Multilayer Network



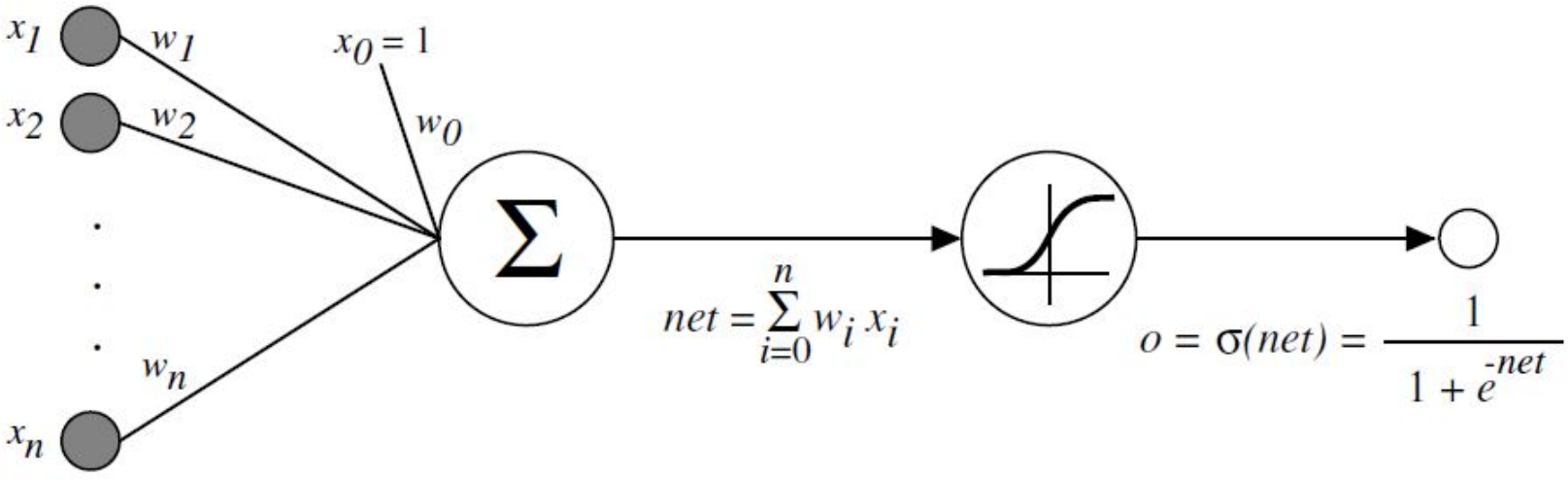
dimana fungsi aktivasi $g(x)$:

- **fungsi threshold** pada *perceptron unit*
- **fungsi identitas** pada *linear unit*
- **fungsi sigmoid** pada *sigmoid unit*

$$\begin{aligned} a_5 &= g(W_{3,5} \times a_3 + W_{4,5} \times a_4) \\ &= g(W_{3,5} \times g(W_{1,3} \times a_1 + W_{2,3} \times a_2) + \\ &\quad W_{4,5} \times g(W_{1,4} \times a_1 + W_{2,4} \times a_2)) \end{aligned}$$

Outline

Sigmoid Unit



- $\sigma(y)$ merupakan fungsi sigmoid: $\frac{1}{1+e^{-y}}$
- Karakteristik fungsi sigmoid: $\frac{d\sigma(y)}{dy} = \sigma(y)(1 - \sigma(y))$
- Kita dapat menurunkan gradient descent untuk melatih
  - satu unit sigmoid
  - multilayer networks dari unit sigmoid → backpropagation

Error Gradient untuk Sebuah Unit Sigmoid

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_d \frac{\partial}{\partial w_i}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_d 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d) \\
&= \sum_d (t_d - o_d)(-\frac{\partial o_d}{\partial w_i}) \\
&= -\sum_d (t_d - o_d)\frac{\partial o_d}{\partial net_d}\frac{\partial net_d}{\partial w_i}
\end{aligned}$$

Tapi, kita mengetahui bahwa

$$\begin{aligned}
\frac{\partial o_d}{\partial net_d} &= \frac{\partial \sigma(net_d)}{\partial net_d} \\
&= o_d(1 - o_d) \\
\frac{\partial net_d}{\partial w_i} &= \frac{\partial(\vec{w}.\vec{x_d})}{\partial w_i} \\
&= x_{i,d}
\end{aligned}$$

Error Gradient untuk Sigmoid Unit

$$\frac{\partial E}{\partial w_i} = -\sum_{d\in D}(t_d - o_d)o_d(1 - o_d)x_{i,d}$$

Outline

Algoritma Backpropagation

- parameter algoritma:
  - *training_examples*: Setiap training example merupakan pasangan dari $\langle \vec{x}, \vec{t} \rangle$, dimana $\vec{x}$ merupakan vektor input dan $\vec{t}$ adalah vektor target output.
  - η merupakan learning rate.
  - n_{in} adalah banyaknya network input, n_{hidden} banyaknya unit di *hidden layer*, dan n_{out} adalah jumlah output unit.
- Buat feed-forward network dengan n_{in} input, n_{hidden} hidden unit, dan n_{out} output unit. Lalu jalankan algoritma backpropagation.

Algoritma Backpropagation

Algoritma Backpropagation

BACKPROPAGATION($training\_examples, \eta, n_{in}, n_{out}, n_{hidden}$)

Inisialisasi setiap nilai $w_{i,j}$ dengan bilangan acak yang bernilai kecil.
Selama kondisi terminasi belum tercapai, lakukan:

- untuk setiap $\langle \vec{x}, t \rangle$ dari training samples, lakukan:
  propagate the inputs forward through the network
  - masukkan $\vec{x}$ ke dalam unit, dan hitung nilai output

  propagate the errors backward through the network
  - Untuk setiap unit output k, hitung errornya δ_k
    $\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$
  - Untuk setiap hidden unit h, hitung errornya δ_h
    $\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k} \delta_k$
  - Update nilai *weight* $w_{i,j}$ pada tiap network:
    $w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$
    dimana $\Delta w_{i,j} = \eta \delta_j x_{i,j}$

Lebih lanjut tentang Backpropagation

- Gradient descent dihitung dari keseluruhan vektor *weight* pada network
- Tidak menjamin mencapai global error minimum
  - Pada praktiknya, seringkali berjalan dengan baik (butuh proses berkali-kali)
- Dapat menambahkan faktor *momentum* α

$$\Delta w_{i,j}(n) = \eta \delta_j x_{i,j} + \alpha \Delta w_{i,j}(n-1)$$

  update weight pada iterasi ke-n bergantung parsial dengan update weight pada iterasi ke-$n-1$
- Meminimumkan error melalui training example
- Proses training dapat melibatkan ribuan iterasi $\rightarrow$ lambat!
- Tapi proses penggunaan setelah training sangat cepat.

Sifat Konvergensi Backpropagation

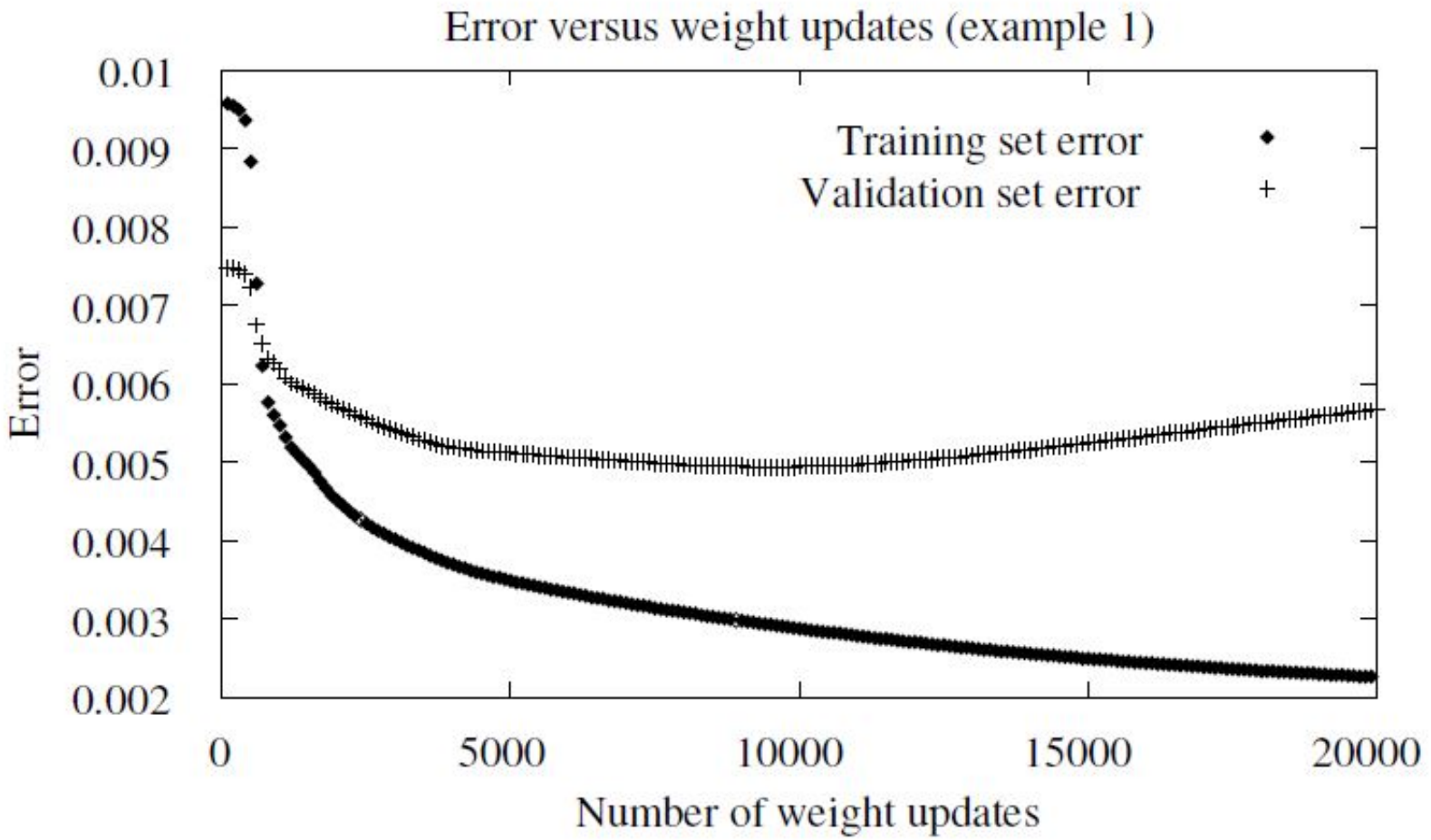
- Gradient descent pada penerapan backpropagation bersifat konvergen menuju local minimum, tapi bisa jadi tidak menuju global minimum
  - tambahkan momentum
  - incremental/stochastic gradient descent
  - latih beberapa network dengan *initial weight* yang berbeda, dan pilih yang performa-nya terbaik.
- Karakteristik konvergensi
  - *weight* diinisialisasi mendekati nol
  - pada awalnya, network akan mengaproksimasi fungsi hampir linear
  - seiring progres training, fungsinya akan menjadi semakin non-linear

Expresiveness ANNs

- Fungsi boolean:
  - Setiap fungsi boolean dapat direpresentasikan oleh network dengan single hidden unit
  - tapi mungkin membutuhkan hidden units yang exponensial banyaknya terhadap input
- Continuous function:
  - Setiap *bounded continuous function* dapat diaproksimasi menggunakan single hidden layer dengan error yang cukup kecil [Cybenko 1989; Hornik et al. 1989].
  - Setiap *continuous function* dapat diaproksimasi dengan akurasi beragam menggunakan network yang terdiri dari dua hidden layer (satu layer dengan sigmoid unit, satu lainnya dengan linear unit) [Cybenko 1988].

Outline

Overfitting pada ANN



Overfitting pada ANN