14620323

DEEP LEARNING

Deep Feedforward Networks

PENGAMPU

Dr. Fajar Astuti Hermawati, S.Kom.,M.Kom.

Bagus Hardiansyah, S.Kom.,M.Si

Andrey Kartika Widhy H., S.Kom., M.Kom.

Capaian Pembelajaran

- **Sub-CPMK-2**: Mampu menyelesaikan masalah komputasi kompleks dengan menerapkan prinsip - prinsip jaringan syaraf tiruan dalam (deep feedforward network) serta regularisasi dan optimisasi pembelajaran dalam pemelajaran mendalam [C 3, A3]

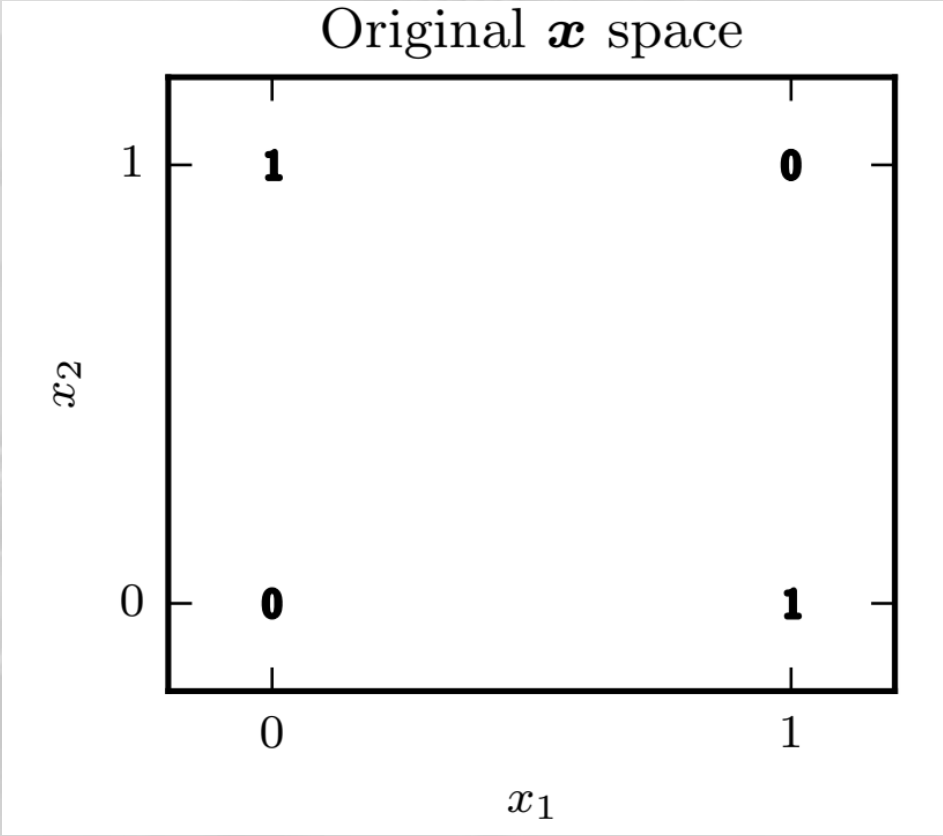
Bahan Kajian

- Example: Learning XOR
- Multilayer Perceptron
- *Interpretability*
- Deep Neural Network
- *Minibatches*

Example: Learning XOR

XOR function

Variabel Input		XOR
x_1	x_2	$x_1 \gg x_2$
0	0	0
0	1	1
1	0	1
1	1	0



XOR function

- Fungsi target $y = \boldsymbol{f}^*(x)$.
- Fungsi model pembelajaran $y = \boldsymbol{f}(x;\vartheta)$, dan algoritma pembelajaran akan mengadaptasi parameter ϑ untuk membuat f semirip mungkin dengan $\boldsymbol{f}^*$
- seluruh rangkaian pembelajaran dievaluai menggunakan , fungsi kerugian (***loss function***) MSE adalah

$$J(\boldsymbol{\theta}) = \frac{1}{4}\sum_{\boldsymbol{x}\in\mathbb{X}} (f^*(\boldsymbol{x}) - f(\boldsymbol{x};\boldsymbol{\theta}))^2 .$$

XOR function

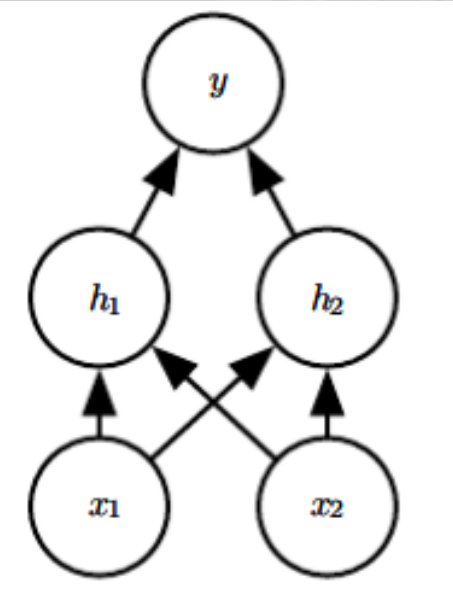
- Misalkan kita memilih model linier, dengan ϑ terdiri dari *w* dan *b*. Model didefinisikan sebagai:

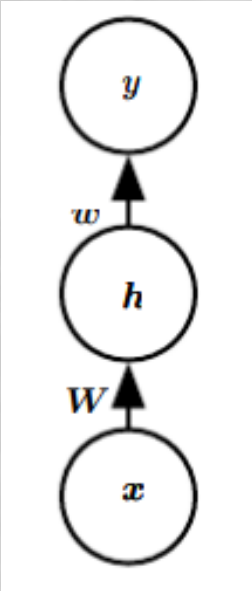
$$f(\boldsymbol{x}; \boldsymbol{w}, b) = \boldsymbol{x}^\top \boldsymbol{w} + b.$$

- Kita dapat meminimalkan $J(\vartheta)$ dalam bentuk tertutup sehubungan dengan *w* dan *b* menggunakan persamaan normal

XOR Network

- Contoh jaringan feedforward, digambar dalam dua gaya berbeda untuk training XOR dengan satu hidden layer.





matriks ***W*** menjelaskan pemetaan dari *x* ke ***h***, dan vektor ***w*** menjelaskan pemetaan dari ***h*** ke *y*.

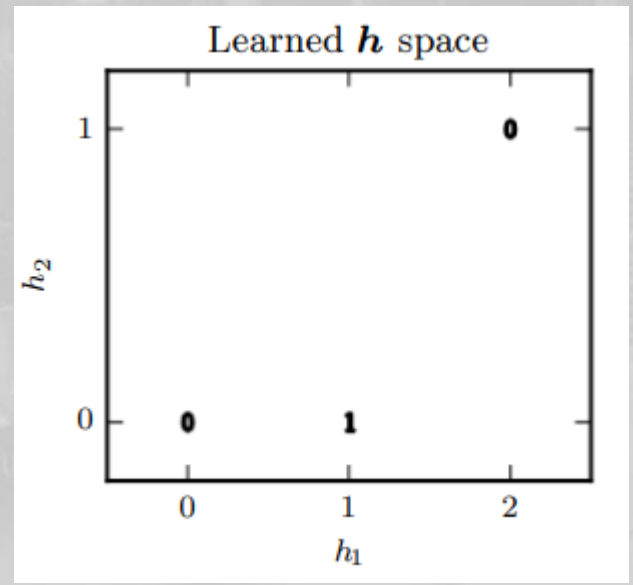
XOR Network

- Jaringan feedforward ini memiliki vektor unit tersembunyi ***h*** yang dihitung dengan fungsi $\boldsymbol{f}^{(1)}(x; \boldsymbol{W}, c)$.
- Nilai unit tersembunyi ini kemudian digunakan sebagai input untuk lapisan kedua. Lapisan kedua adalah lapisan keluaran jaringan (***output layer***).
- Lapisan keluaran masih berupa model regresi linier, tetapi sekarang diterapkan ke ***h*** daripada ke ***x***.

XOR Network

- Jaringan sekarang berisi dua fungsi yang dirangkai bersama, $\boldsymbol{h} = f^{(1)}(\boldsymbol{x}; \boldsymbol{W}, c)$ dan $y = f^{(2)}(\boldsymbol{h}; \boldsymbol{w}, b)$, dengan model lengkapnya adalah

$$f(\boldsymbol{x}; \boldsymbol{W}, c, \boldsymbol{w}, b) = f^{(2)}(f^{(1)}(\boldsymbol{x})).$$



XOR Function Computation

- Jika $f^{(1)}(x) = \boldsymbol{W}^{\mathsf{T}}x$ and $f^{(2)}(\boldsymbol{h}) = \boldsymbol{h}^{\mathsf{T}}\boldsymbol{w}$. Maka $f(\boldsymbol{x}) = \boldsymbol{x}^{\mathsf{T}}\boldsymbol{W}\boldsymbol{w}$.
- Kita dapat menyatakan fungsi ini sebagai

  $f(\boldsymbol{x}) = \boldsymbol{x}^{\mathsf{T}}\boldsymbol{w}'$ dimana $\boldsymbol{w}' = \boldsymbol{W}\boldsymbol{w}$.

Activation Function

- Kita harus menggunakan fungsi **nonlinear** untuk mendeskripsikan fitur.
- Sebagian besar jaringan saraf melakukannya dengan menggunakan **transformasi *affine*** yang dikontrol oleh parameter yang dipelajari, diikuti oleh fungsi nonlinier tetap yang disebut ***fungsi aktivasi***.
- Kita menggunakan strategi **transformasi *affine***, dengan mendefinisikan

$$\boldsymbol{h} = g(\boldsymbol{W}x + c),$$

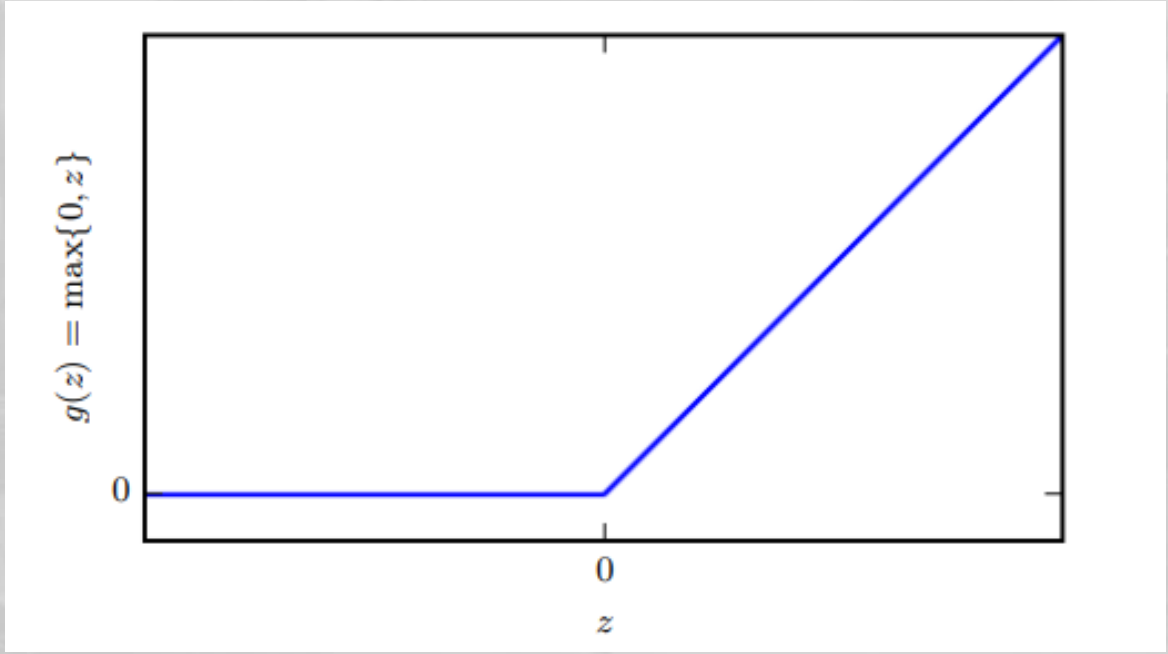
- di mana $\boldsymbol{W}$ memberikan bobot transformasi linier dan $\boldsymbol{c}$ bias.

Activation Function

- Sekarang, kita menggambarkan transformasi ***affine*** dari vektor ***x*** ke vektor ***h***, sehingga diperlukan seluruh vektor parameter bias.
- Fungsi aktivasi *g* biasanya dipilih sebagai fungsi yang diterapkan berdasarkan elemen, dengan $h_i = g(\boldsymbol{x}^T \boldsymbol{W}_{:,i} + c_i)$.
- Dalam jaringan saraf modern, rekomendasi default adalah menggunakan unit linier yang diperbaiki (***rectified linear unit***), atau **ReLU** (Jarrett et al., 2009; Nair dan Hinton, 2010; Glorot et al., 2011a), ditentukan oleh fungsi aktivasi $\boldsymbol{g}(z) = \text{maks}\{0, z\}$

Activation Function

- Menerapkan fungsi ***rectified linear unit*** ke keluaran transformasi linier menghasilkan transformasi nonlinier.
- Fungsinya tetap sangat dekat dengan linier, dalam arti bahwa itu adalah fungsi linier sepotong-sepotong dengan dua bagian linier.
- Karena unit linier terkoreksi hampir linier, mereka mempertahankan banyak properti yang membuat model linier mudah dioptimalkan dengan metode berbasis gradien.



Feedforward computation

- Jaringan lengkap kita dapat dinyatakan sebagai:

$$f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^\top \max\{0, \boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c}\} + b.$$

- Kita dapat menentukan solusi untuk masalah XOR. Dimana misalkan b=0 dan

$$\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix},$$

$$\boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix},$$

$$\boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix},$$

Feedforward computation

- Kita sekarang dapat menelusuri bagaimana model memproses sekumpulan input.
- Misalkan $\boldsymbol{X}$ adalah matriks desain yang memuat keempat titik dalam ruang masukan biner, dengan satu contoh per baris

$$\boldsymbol{X} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}.$$

Feedforward computation

- Langkah pertama dalam jaringan saraf adalah mengalikan matriks input dengan matriks bobot lapisan pertama:

$$\boldsymbol{XW} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}.$$

- Selanjutnya, kita tambahkan vektor bias *c*, untuk mendapatkan

$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}.$$

Feedforward computation

- Untuk menyelesaikan penghitungan nilai ***h*** untuk setiap contoh, diterapkan transformasi ***rectified linear***

$$\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}.$$

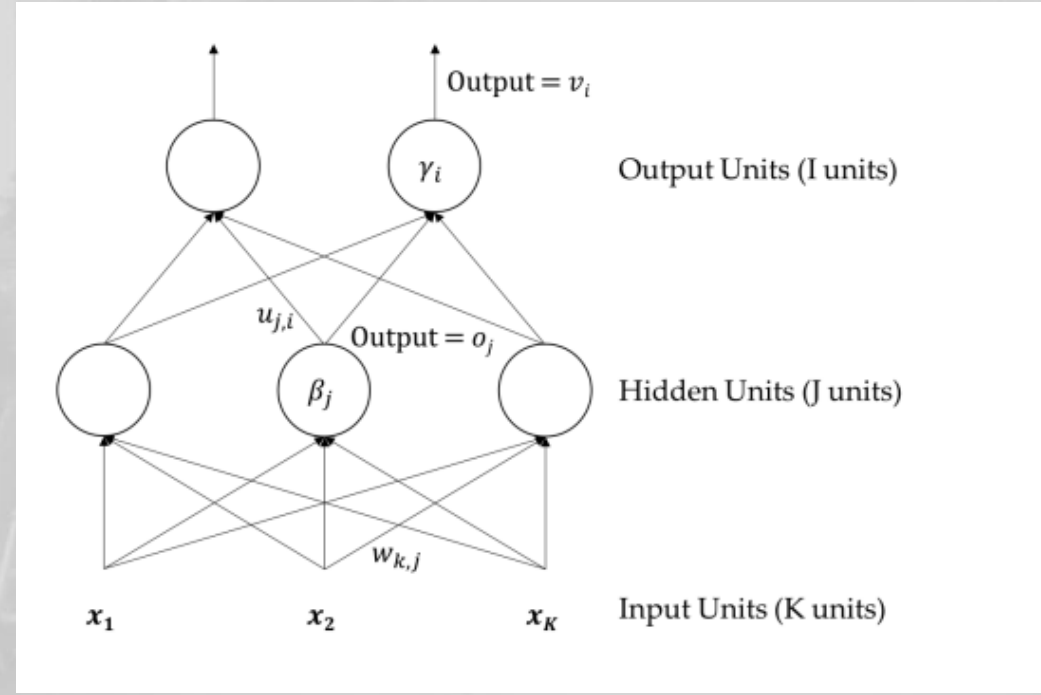
- Kita akhiri dengan mengalikan dengan vektor bobot ***w***:

$$\begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$

Multilayer Perceptron

multilayer perceptron (MLP)

- multilayer perceptron (MLP) yang juga dikenal sebagai feedforward neural network.
- secara umum ada tiga layers: input, hidden, dan output layer.
- ***u***, ***w*** adalah learning parameters.
- α,β melambangkan noise atau bias
- ***K*** adalah banyaknya input units dan ***J*** adalah banyaknya hidden units



multilayer perceptron (MLP)

- Input layer menerima input (tanpa melakukan operasi apapun), kemudian nilai input (tanpa dilewatkan ke fungsi aktivasi) diberikan ke hidden units:

$$o_j = \sigma\left(\sum_{k=1}^{K} x_k w_{k,j} + \beta_j\right)$$

- Pada hidden units, input diproses dan dilakukan perhitungan hasil fungsi aktivasi untuk tiap-tiap neuron, lalu hasilnya diberikan ke layer berikutnya (persamaan berikut adalah fungsi aktivasi).

$$y_i = \sigma\left(\sum_{j=1}^{J} o_j u_{j,i} + \gamma_i\right) = \sigma\left(\sum_{j=1}^{J} \sigma\left(\sum_{k=1}^{K} x_k w_{k,j} + \beta_j\right) u_{j,i} + \gamma_i\right)$$

multilayer perceptron (MLP)

- Output dari input layer akan diterima sebagai input bagi hidden layer. Begitupula seterusnya hidden layer akan mengirimkan hasilnya untuk output layer.
- Kegiatan ini dinamakan ***feed forward***.
- Persamaan fungsi aktivasi dapat disederhanakan dengan:

$$\mathbf{v} = \sigma(\mathbf{o}\mathbf{U} + \gamma) = \sigma((\sigma(\mathbf{x}\mathbf{W} + \beta))\mathbf{U} + \gamma)$$

multilayer perceptron (MLP)

- Hal serupa berlaku untuk artificial neural network dengan lebih dari tiga layers.
- Parameter neuron dapat dioptimisasi menggunakan ***metode gradient-based optimization***.
- Perlu diperhatikan, MLP adalah gabungan dari banyak fungsi non-linear.
- Masing-masing neuron terkoneksi dengan semua neuron pada layer berikutnya.
- Konfigurasi ini disebut sebagai ***fully connected***.

back propagation.

- Untuk melatih MLP, algoritma yang umumnya digunakan adalah ***back propagation***.
  - Memperbaharui parameter (*synapse weights*) secara bertahap (dari output ke input layer, karena itu disebut backpropagation) berdasarkan ***error/loss*** (output dibandingkan dengan desired output).
  - Intinya adalah mengkoreksi ***synapse weight*** dari output layer ke hidden layer, kemudian ***error*** tersebut dipropagasi ke layer sebelum-sebelumnya.
  - Artinya, perubahan ***synapse weight*** pada suatu layer dipengaruhi oleh perubahan ***synapse weight*** pada layer setelahnya.
- Backpropagation adalah ***metode gradient-based optimization*** yang diterapkan pada ANN

back propagation.

- Pertama-tama diberikan pasangan input (*x*) dan desired output (*y*) sebagai training data.
- Untuk **meminimalkan *loss***, algoritma backpropagation menggunakan prinsip ***gradient descent***.
- Error, untuk MLP diberikan oleh persamaan mean squared error

$$E(\theta) = \frac{1}{2}\sum_{i=1}^{I}(y_i - v_i)^2$$

- dimana *I* adalah banyaknya output unit dan θ adalah kumpulan ***weight matrices*** (semua parameter pada MLP).

back propagation

- Proses penurunan untuk melatih MLP.
- Error/loss diturunkan terhadap tiap learning parameter.
- Diferensial $\boldsymbol{u}_{j,i}$ diberikan oleh turunan sigmoid function

$$\frac{\delta E(\theta)}{\delta u_{j,i}} = (y_i - v_i)\frac{\delta v_i}{\delta u_{j,i}}$$
$$= (y_i - v_i)v_i(1 - v_i)o_j$$

back propagation

- Diferensial $w_{k,j}$ diberikan oleh turunan sigmoid function

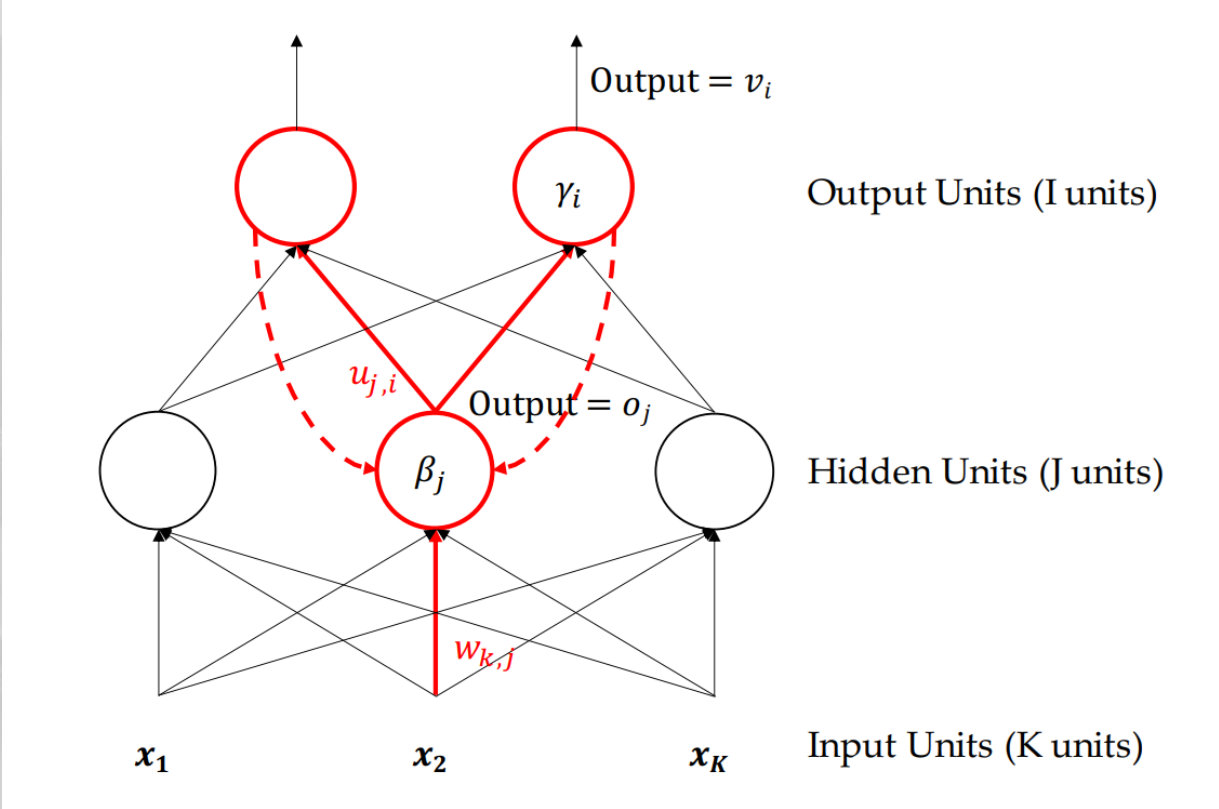
$$\frac{\delta E(\theta)}{\delta w_{k,j}} = \sum_{i=1}^{I} (y_i - v_i) \frac{\delta v_i}{\delta w_{k,j}}$$

$$= \sum_{i=1}^{I} (y_i - v_i) \frac{\delta v_i}{\delta o_j} \frac{\delta o_j}{\delta w_{k,j}}$$

$$= \sum_{i=1}^{I} (y_i - v_i)(v_i(1 - v_i)u_{j,i})(o_j(1 - o_j)x_k)$$

back propagation

- Perhatikan, diferensial $w_{k,j}$ memiliki ∑ sementara $u_{j,i}$ tidak ada.
- Hal ini disebabkan karena $u_{j,i}$ hanya berkorespondensi dengan satu output neuron. Sementara $w_{k,j}$ berkorespondensi dengan banyak output neuron.
- Dengan kata lain, nilai $w_{k,j}$ mempengaruhi hasil operasi yang terjadi pada banyak output neuron, sehingga banyak neuron mempropagasi error kembali ke $w_{k,j}$



back propagation

- Proses latihan MLP menggunakan backpropagation

(2) Hidden to Output

$$v_i = \sigma\left(\sum_{j=1}^{J} o_j u_{j,i} + \gamma_i\right)$$

(3) Output to Hidden

$$\delta_i = (y_i - v_i)v_i(1 - v_i)$$
$$\Delta u_{j,i} = -\eta(t)\delta_i o_j$$
$$\Delta \gamma_i = -\eta(t)\delta_i$$

(1) Input to Hidden Layer

$$o_j = \sigma\left(\sum_{k=1}^{K} x_k w_{k,j} + \beta_j\right)$$

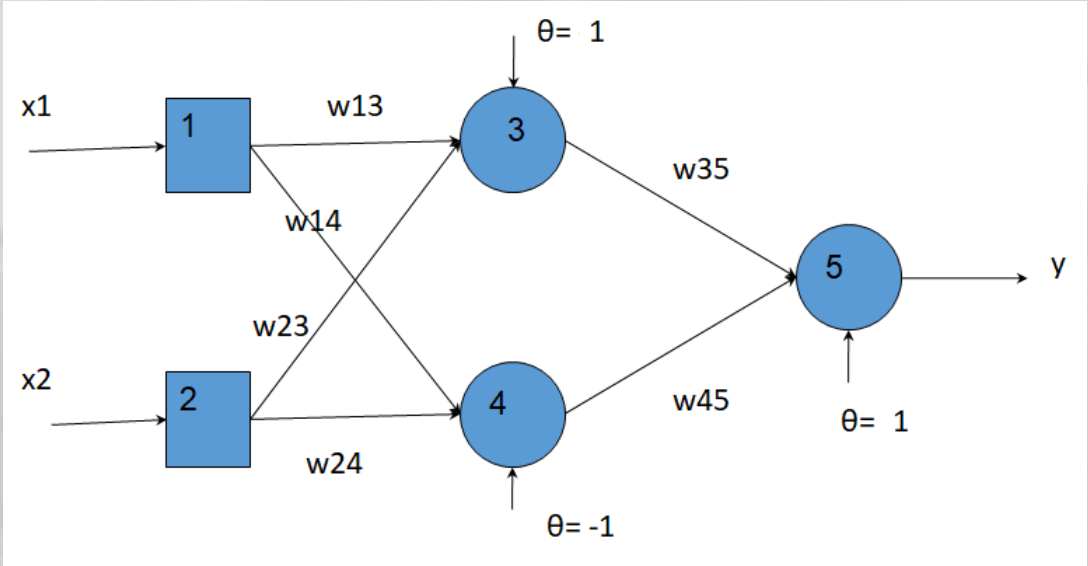
(4) Hidden to Input

$$\varphi_j = \sum_{i=1}^{I} \delta_i u_{j,i} o_j(1 - o_j)$$
$$\Delta w_{k,j} = -\eta(t)\varphi_j x_k$$
$$\Delta \beta_j = -\eta(t)\varphi_j$$

Contoh Perhitungan 1 iterasi permasalahan XOR

- Diketahui :
  - α = 0,1
  - Threshold neuron 3 = Θ3 = 1
  - Threshold neuron 4 = Θ4 = -1
  - Threshold neuron 5 = Θ5 = 1
  - Fungsi aktivasi = relu

$$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}.$$



- Initial weight untuk ketiga neuron :
  - Hidden Neuron 3 : w13 = 0,1; w23 = 0,3
  - Hidden Neuron 4 : w14 = -0,1; w24 = 0,5
  - Output Neuron 5 : w35 = -0.9; w45 = 0,1

$$W=\begin{bmatrix} 0.1 & -0.1 \\ 0.3 & 0.5 \end{bmatrix} \; w = \begin{bmatrix} -0.9 \\ 0.1 \end{bmatrix} \beta = c = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \gamma=b=1 \; ; y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Contoh Perhitungan 1 iterasi

- Input to hidden layer (feed forward):

$$O = \sigma(X.W + \beta) = \sigma\left(\begin{bmatrix}0 & 0\\ 0 & 1\\ 1 & 0\\ 1 & 1\end{bmatrix}\begin{bmatrix}0.1 & -0.1\\ 0.3 & 0.5\end{bmatrix} + \begin{bmatrix}1 & -1\end{bmatrix}\right) = \sigma\left(\begin{bmatrix} 0 & 0\\ 0.3 & 0.5\\ 0.1 & -0.1\\ 0.4 & 0.4\end{bmatrix}\right.$$

$$\left. + \begin{bmatrix}1 & -1\\ 1 & -1\\ 1 & -1\\ 1 & -1\end{bmatrix}\right) = \sigma\left(\begin{bmatrix} 1 & -1\\ 1.3 & -0.5\\ 1.1 & -1.1\\ 1.4 & -0.6\end{bmatrix}\right) = \begin{bmatrix}1 & 0\\ 1.3 & 0\\ 1.1 & 0\\ 1.4 & 0\end{bmatrix}$$

Contoh Perhitungan 1 iterasi

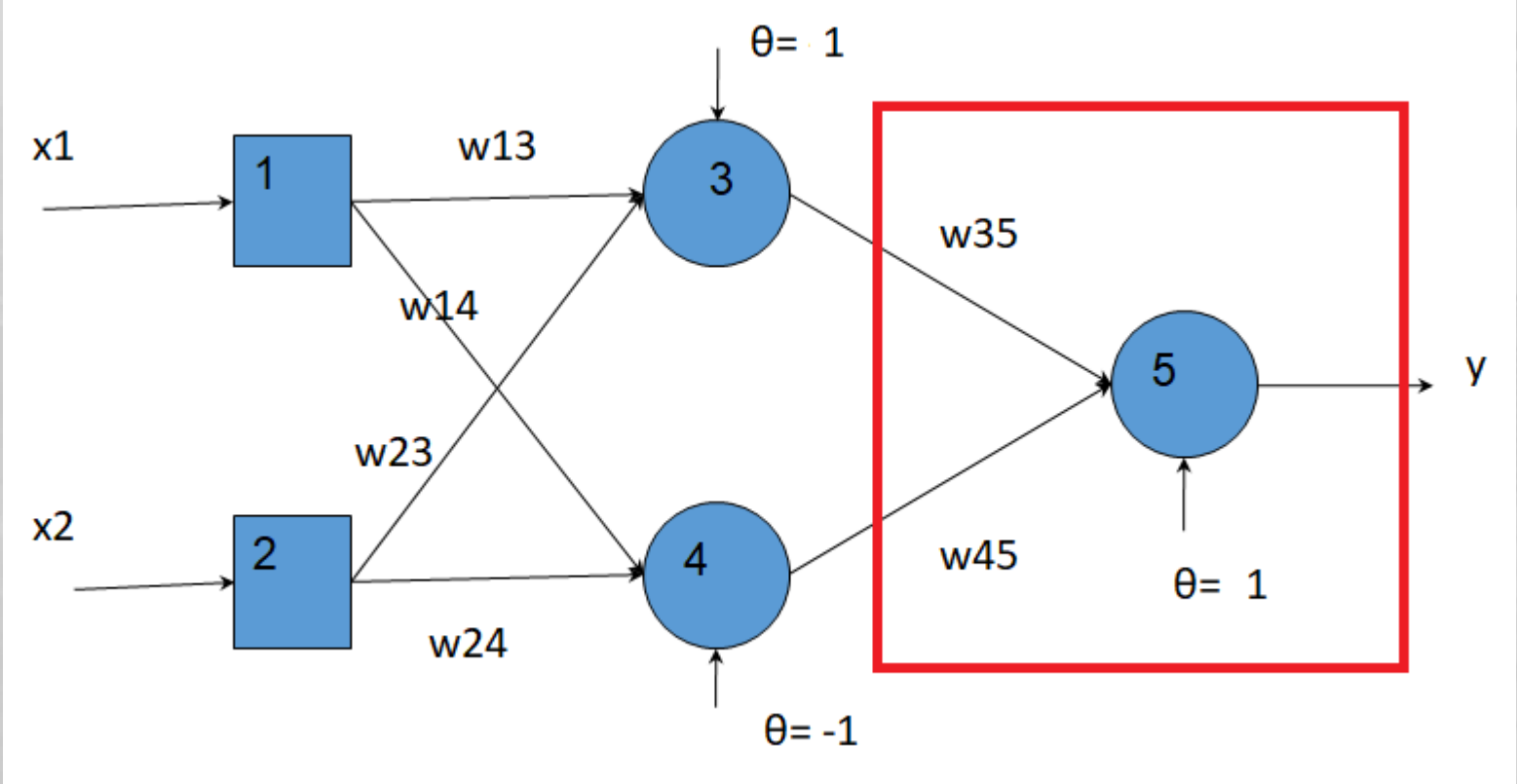
- hidden layer to output (feed forward):

$$v = \sigma(o.w + \gamma) = \sigma\left(\begin{bmatrix} 1 & 0 \\ 1.3 & 0 \\ 1.1 & 0 \\ 1.4 & 0 \end{bmatrix}\begin{bmatrix} -0.9 \\ 0.1 \end{bmatrix} + [1]\right) = \sigma\left(\begin{bmatrix} -0.9 \\ 1.17 \\ 0.99 \\ 1.26 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}\right)$$

$$= \sigma\left(\begin{bmatrix} 0.1 \\ 2.17 \\ 1.99 \\ 2.26 \end{bmatrix}\right) = \begin{bmatrix} 0.1 \\ 2.17 \\ 1.99 \\ 2.26 \end{bmatrix}$$

Contoh Perhitungan 1 iterasi

- Training Bobot Output Layer



Contoh Perhitungan 1 iterasi

- Output to hidden (back propagation):

$$\delta = (y - v)v^T(1 - v) = \left(\begin{bmatrix}0\\1\\1\\0\end{bmatrix} - \begin{bmatrix}0.1\\1.17\\0.99\\1.26\end{bmatrix}\right)[0.1\ 1.17\ 0.99\ 1.26]\left(\begin{bmatrix}1\\1\\1\\1\end{bmatrix} - \begin{bmatrix}0.1\\1.17\\0.99\\1.26\end{bmatrix}\right)$$

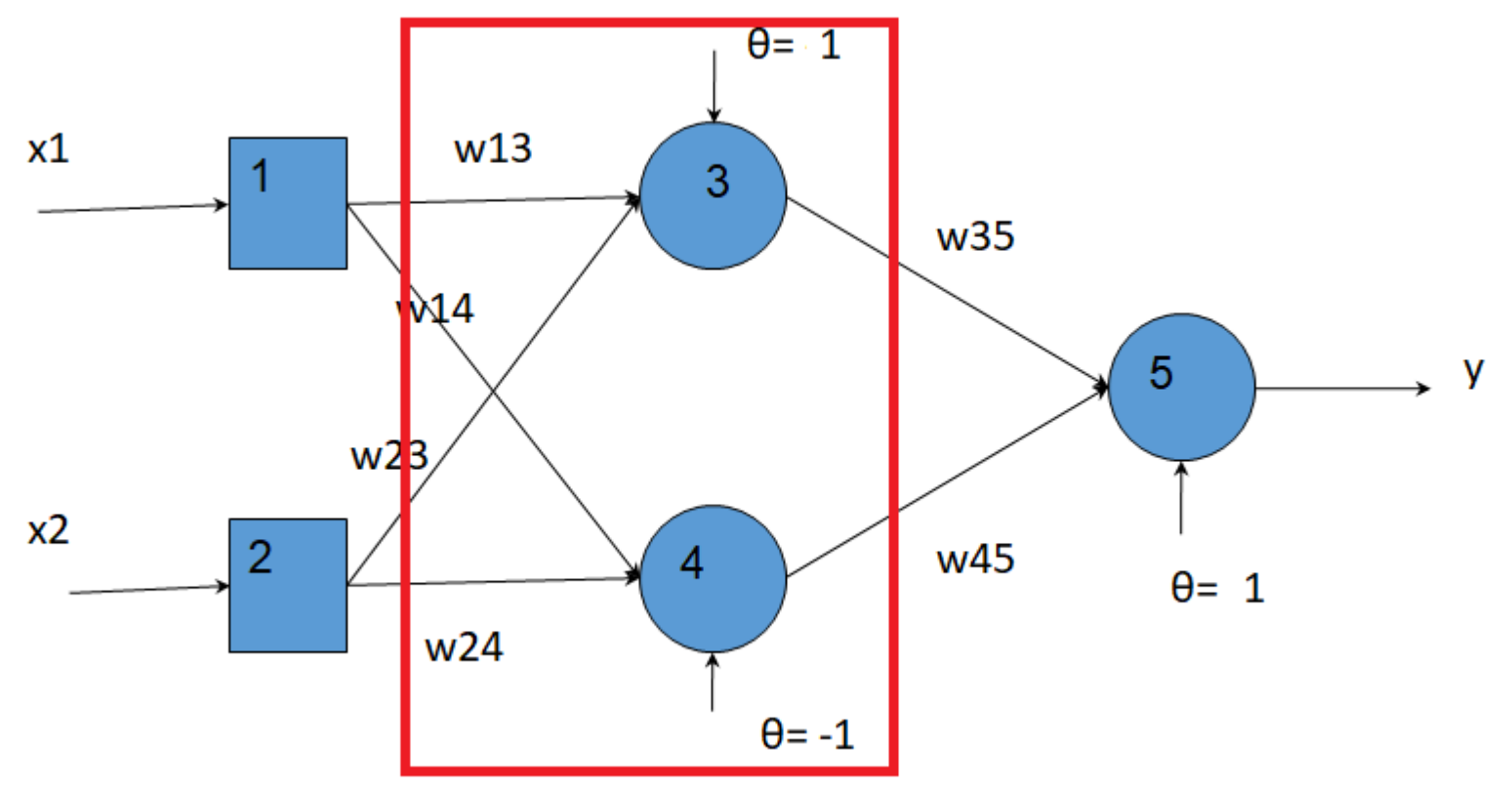
$$= \begin{bmatrix}-0.1\\-0.17\\0.01\\-1.26\end{bmatrix}[0.1\ 1.17\ 0.99\ 1.26]\left(\begin{bmatrix}0.9\\-0.17\\0.01\\-0.26\end{bmatrix}\right) = \begin{bmatrix}-0.1\\-0.17\\0.01\\-1.26\end{bmatrix}(0.09 - 0{,}1989 + 0.0099 - 0.3276) = \begin{bmatrix}-0.1\\-0.17\\0.01\\-1.26\end{bmatrix}(-0.43) = \begin{bmatrix}0.043\\0.073\\-0.0043\\0.54\end{bmatrix}$$

$$\Delta w = -\alpha\delta^T o = -0.1[0.043 \quad 0.073 \quad -0.0043 \quad 0.54]\begin{bmatrix}1 & 0\\1.3 & 0\\1.1 & 0\\1.4 & 0\end{bmatrix} = -0.1[0.89 \quad 0] = [-0.089 \quad 0]$$

$$w' = w + \Delta w = \begin{bmatrix}-0.9\\0.1\end{bmatrix} + \begin{bmatrix}-0.089\\0\end{bmatrix} = \begin{bmatrix}-0.98\\0.1\end{bmatrix}$$

Contoh Perhitungan 1 iterasi

- Training Bobot Hidden Layer



(4) Hidden to Input

$$\varphi_j = \sum_{i=1}^{I} \delta_i u_{j,i} o_j (1 - o_j)$$

$$\Delta w_{k,j} = -\eta(t)\varphi_j x_k$$

$$\Delta \beta_j = -\eta(t)\varphi_j$$

Contoh Perhitungan 1 iterasi

- Hidden to input (*back propagation*):

$$\varphi = \delta u^T o^T (1 - o) = \begin{bmatrix} 0.043 \\ 0.073 \\ -0.0043 \\ 0.54 \end{bmatrix} [[-0.98 \;\; 0.1\,]] \begin{bmatrix} 1 & 1.3 & 1.1 & 1.4 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ -0.3 & 0 \\ -0.1 & 0 \\ -0.4 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0.043 \\ 0.073 \\ -0.0043 \\ 0.54 \end{bmatrix} [[-0.98 \;\; 0.1\,]] \begin{bmatrix} -1.06 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0.043 \\ 0.073 \\ -0.0043 \\ 0.54 \end{bmatrix} [1.04 \quad 0] = \begin{bmatrix} 0.045 & 0 \\ 0.076 & 0 \\ -0.0045 & 0 \\ 0.56 & 0 \end{bmatrix}$$

$$\Delta w = -\alpha \varphi^T x = -0.1 \begin{bmatrix} 0.045 & 0.076 & -0.0045 & 0.56 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} = -0.1 \begin{bmatrix} 0.557 & 0.636 \\ 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} -0.0557 & -0.0636 \\ 0 & 0 \end{bmatrix}$$

$$w' = w + \Delta w = \begin{bmatrix} 0.1 & -0.1 \\ 0.3 & 0.5 \end{bmatrix} + \begin{bmatrix} -0.0557 & -0.0636 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0.09 & -0.16 \\ 0.3 & 0.5 \end{bmatrix}$$

Contoh Perhitungan 1 iterasi (1 epoch)

- Weight yang diperoleh untuk ketiga neuron :
  - Hidden Neuron 3 : w13 = 0.619; w23 = 0,3
  - Hidden Neuron 4 : w14 = 0.495; w24 = 0,5
  - Output Neuron 5 : w35 = -0.98; w45 = 0,1
- Dengan menerapkan bobot yang diperoleh pada pelatihan maka kita dapat menghitung luaran keseluruhan dengan menggunakan perhitungan feed forward seperti diatas

Interpretability

Interpretability

- *Interpretability* ada dua macam yaitu:
  - ***model interpretability*** (i.e., apakah struktur model pembelajaran mesin dapat dipahami) dan
  - ***prediction interpretability*** (i.e., bagaimana memahami dan memverifika cara input dipetakan menjadi output)
- ANN (MLP) biasanya dianggap sebagai metode black box atau susah untuk dinterpretasikan (terutama model interpretability-nya).
- Hal ini disebabkan oleh kedalaman (*depth*) yaitu memiliki beberapa layer dan non-linearities.

Interpretability

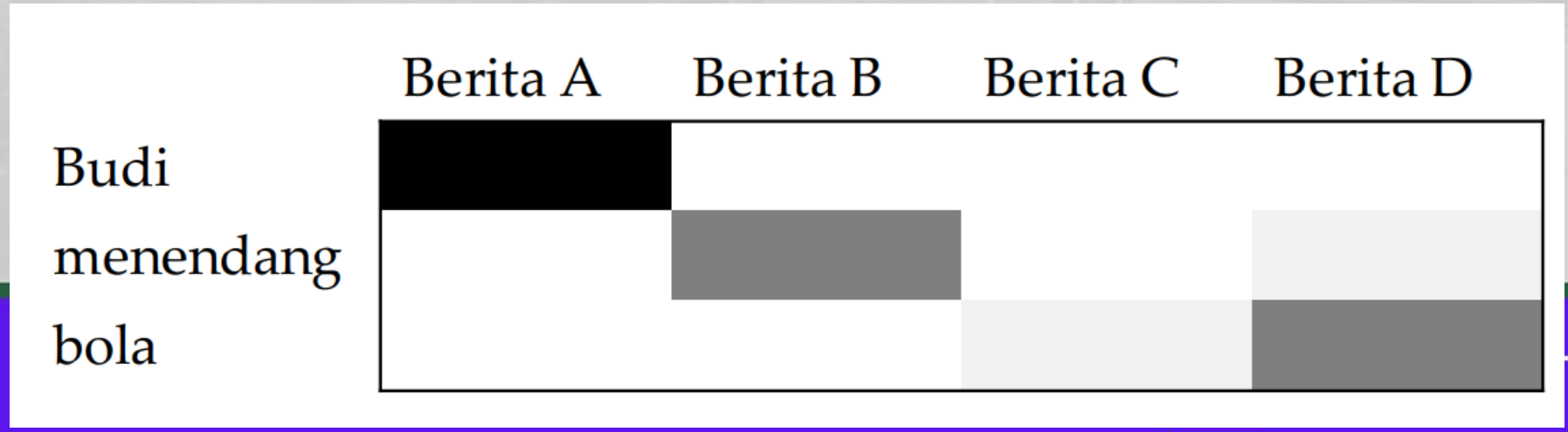
- Suatu unit pada output layer dipengaruhi oleh kombinasi (*arbitrary combination*) banyak parameter pada layers sebelumnya yang dilewatkan pada fungsi non-linear
- Intinya, **fitur dan output tidak memiliki korespondensi satu-satu**. Berbeda dengan model linear, kita tahu parameter (dan bobotnya) untuk setiap input.

Interpretability

- Cara paling umum untuk menjelaskan keputusan pada ANN adalah menggunakan ***heat map***.
- Sederhananya, kita lewatkan suatu data ***x*** pada ANN, kemudian kita lakukan *feed-forward* sekali (misal dari input ke hidden layer dengan parameter ***W***).
- Kemudian, kita visualisasikan $\boldsymbol{x} \cdot \boldsymbol{W}$
- Dengan ini, kita kurang lebih dapat mengetahui bagian input mana yang berpengaruh terhadap keputusan di layer berikutnya.

Interpretability

- Contoh heat map pada persoalan klasifikasi teks.
  - Input berupa kata-kata yang dimuat pada suatu berita.
  - Output adalah kategori berita untuk input.
  - Warna lebih gelap menandakan bobot lebih tinggi.
  - Sebagai contoh, kata “menendang” berkorespondensi paling erat dengan kelas berita B



Binary Classification

- Salah satu strategi untuk ***binary classification*** adalah dengan menyediakan hanya satu output unit di jaringan.
- Kelas pertama direpresentasikan dengan -1
- kelas kedua direpresentasikan dengan nilai 1 (setelah diaktivasi)
- Hal ini dapat dicapai dengan fungsi non-linear seperti **sign** atau **tanh**.
- Kita dapat menggunakan fungsi seperti sigmoid, dimana output pada masing-masing neuron berada pada range nilai [0, 1], jika kita ingin mengukur probabilitas yang masuk ke sebuah kelas

Multi-class Classification

- Multilayer perceptron dapat memiliki lebih dari satu output unit.
- Seumpama kita mempunyai empat kelas, dengan demikian kita dapat merepresentasikan keempat kelas tersebut sebagai empat output units. Kelas pertama direpresentasikan dengan unit pertama, kelas kedua dengan unit kedua, dst.
- Untuk ***C*** kelas, kita dapat merepresentasikannya dengan ***C*** output units.

Multi-class Classification

- Kita dapat merepresentasikan data harus dimasukkan ke kelas mana menggunakan ***sparse vector***, yaitu bernilai **0** atau **1**.
- Elemen ke-*i* bernilai **1** apabila data masuk ke kelas c_i, sementara nilai elemen lainnya adalah 0

$$\begin{matrix} c_1 & c_2 & c_3 & c_4 \end{matrix}$$
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{matrix} \text{Label} = c_1 \\ \text{Label} = c_2 \\ \text{Label} = c_3 \\ \text{Label} = c_4 \end{matrix}$$

Multi-class Classification

- Output ANN dilewatkan pada suatu fungsi ***softmax*** yang melambangkan probabilitas ***class-assignment***; i.e., **kita ingin output agar semirip mungkin dengan sparse vector (*desired output*).**
- Pada kasus ini, output ANN adalah sebuah distribusi yang melambangkan input di-assign ke kelas tertentu.

Multi-label Classification

- Seperti halnya multi-class classification, kita dapat menggunakan sebanyak ***C*** neuron untuk merepresentasikan ***C*** kelas pada multi-label classification.
- Perbedaan **multi-class dan multilabel** terletak pada cara interpretasi output dan evaluasi output.
- Pada umumnya, layer terakhir diaktivasi dengan fungsi sigmoid, dimana tiap neuron $\boldsymbol{n}_i$ merepresentasikan probabilitas suatu dapat diklasifikasikan sebagai kelas $\boldsymbol{c}_i$ atau tidak

Multi-label Classification

- Ilustrasi representasi desired output pada multi-label classification.

$$\begin{matrix} c_1 & c_2 & c_3 & c_4 \end{matrix}$$

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

$\text{Label} = c_1, c_3$

$\text{Label} = c_2$

$\text{Label} = c_1, c_4$

$\text{Label} = c_2, c_3, c_4$

- Pada umumnya, ***binary cross entropy*** digunakan sebagai ***loss*** (utility function) pada multi-label classification, yaitu perhitungan ***cross entropy*** untuk tiap-tiap output unit (bukan sekaligus semua output unit seperti pada multi-class classification).

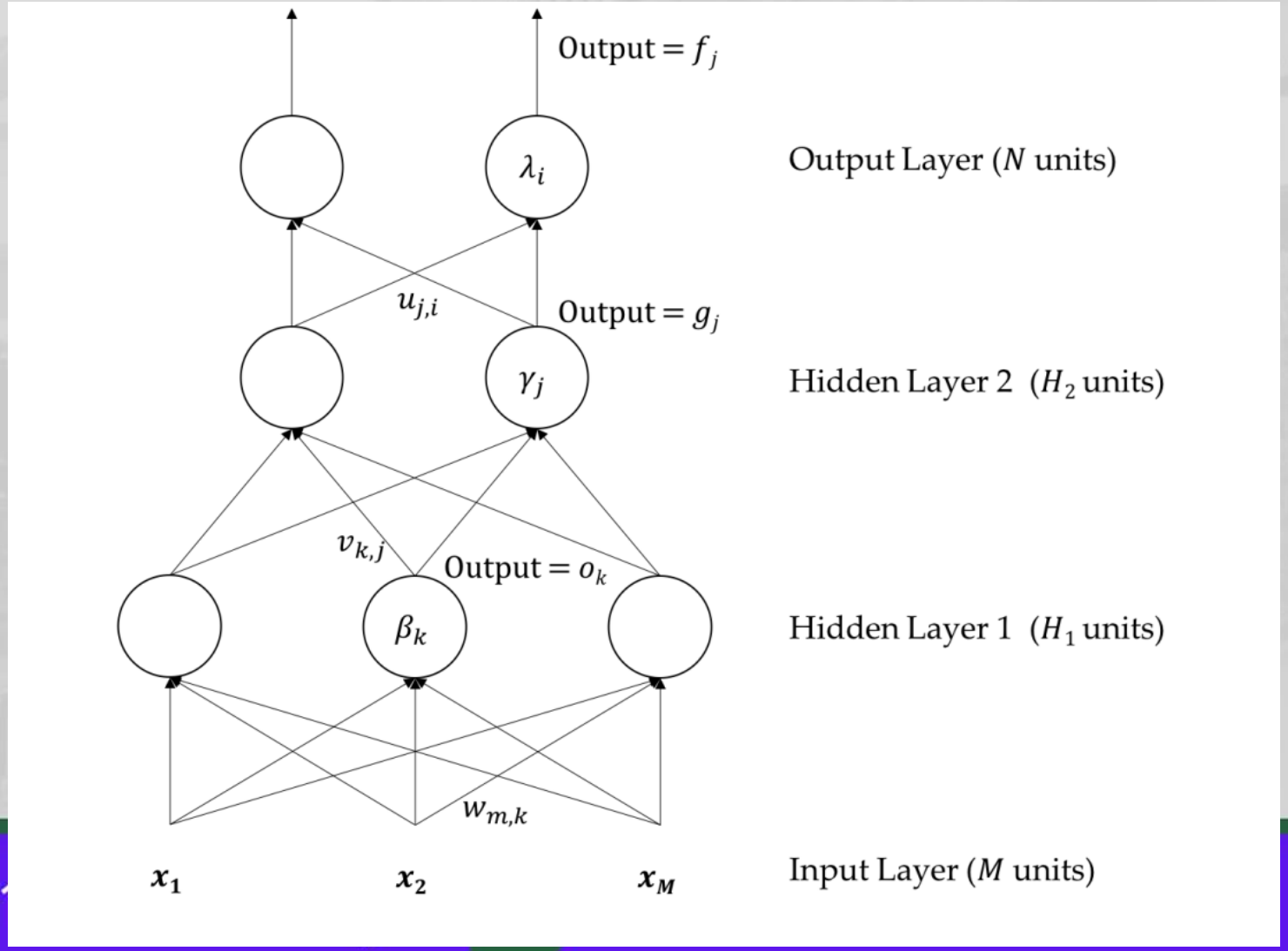
Deep Neural Network

Deep Neural Network

- **Deep Neural Network (DNN)** adalah artificial neural network yang memiliki **banyak layer**.
- Pada umumnya, deep neural network memiliki lebih dari 3 layers (input layer, N ≥2 hidden layers, output layer), dengan kata lain adalah MLP dengan lebih banyak layer.
- Karena ada relatif banyak layer, disebutlah ***deep***.
- Proses pembelajaran pada DNN disebut sebagai deep learning (tidak ada bedanya dengan proses latihan pada jaringan yang lebih dangkal)
- **Jaringan neural network pada DNN disebut deep neural network**.

Deep Neural Network

- Contoh **deep neural network** yang memiliki 4 layers



Deep Neural Network

- Cara menghitung final output sama seperti MLP, diberikan pada persamaan

$$f_i = \sigma\left(\sum_{j=1}^{H_2} u_{j,i}\sigma\left(\sum_{k=1}^{H_1} v_{k,j}\sigma\left(\sum_{m=1}^{M} x_m w_{m,k} + \beta_k\right) + \gamma_j\right) + \lambda_i\right)$$

- α,β, γ adalah noise atau bias, σ adalah fungsi aktivasi.

Deep Neural Network

- Cara melatih deep neural network, salah satunya dapat menggunakan backpropagation.
- Seperti pada bagian sebelumnya, kita hanya perlu menurunkan rumusnya saja

(3) Hidden 2 to Output

$$f_i = \sigma\left(\sum_{j=1}^{H_2} g_j u_{j,i} + \lambda_i\right)$$

(2) Hidden 1 to Hidden 2

$$g_j = \sigma\left(\sum_{k=1}^{H_1} o_k v_{k,j} + \gamma_j\right)$$

(1) Input to Hidden Layer

$$o_k = \sigma\left(\sum_{m=1}^{M} x_m w_{m,k} + \beta_k\right)$$

(4) Output to Hidden 2

$$\delta_i = (y_i - f_i) f_i (1 - f_i)$$
$$\Delta u_{j,i} = -\eta(t)\delta_i g_j$$
$$\Delta\lambda_i = -\eta(t)\delta_i$$

(5) Hidden 2 to Hidden 1

$$\varphi_j = \sum_{i=1}^{N} \delta_i u_{j,i} g_j (1 - g_j)$$
$$\Delta v_{k,j} = -\eta(t)\varphi_j o_k$$
$$\Delta\gamma_j = -\eta(t)\varphi_j$$

(6) Hidden 1 to Input

$$\mu_k = \sum_{j=1}^{H_2} \varphi_j v_{k,j} o_k (1 - o_k)$$
$$\Delta w_{m,k} = -\eta(t)\mu_k x_m$$
$$\Delta\beta_k = -\eta(t)\beta_k$$

Deep Neural Network

- Deep network terdiri dari **banyak layer dan *synapse weight***, karenanya estimasi parameter susah dilakukan.
- Arti filosofisnya adalah susah/lama untuk menentukan relasi antara input dan output.
- Walaupun deep learning sepertinya kompleks, tetapi dapat bekerja dengan baik untuk permasalahan praktis.
- Deep learning dapat menemukan relasi “tersembunyi” antara input dan output, yang tidak dapat diselesaikan menggunakan multilayer perceptron (3 layers).

Deep Neural Network

- Perhatikan, harus ingat bahwa **satu langkah feedforward memiliki analogi dengan transformasi.**
- Jadi, input ditransformasikan secara non-linear sampai akhirnya pada output, berbentuk ***distribusi class-assignment***.
- Banyak orang percaya deep neural network lebih baik dibanding neural network yang lebar tapi sedikit layer, karena terjadi lebih **banyak transformasi**.

Deep Neural Network

- Maksud lebih **banyak transformasi** adalah **kemampuan untuk merubah input menjadi suatu representasi** (tiap hidden layer dapat dianggap sebagai salah satu bentuk representasi input) dengan langkah hierarchical.
- Seperti contoh permasalahan XOR, permasalahan non-linearly separable pun dapat diselesaikan apabila kita dapat mentransformasi data (representasi data) ke dalam bentuk linearly separable pada ruang yang berbeda.

Deep Neural Network

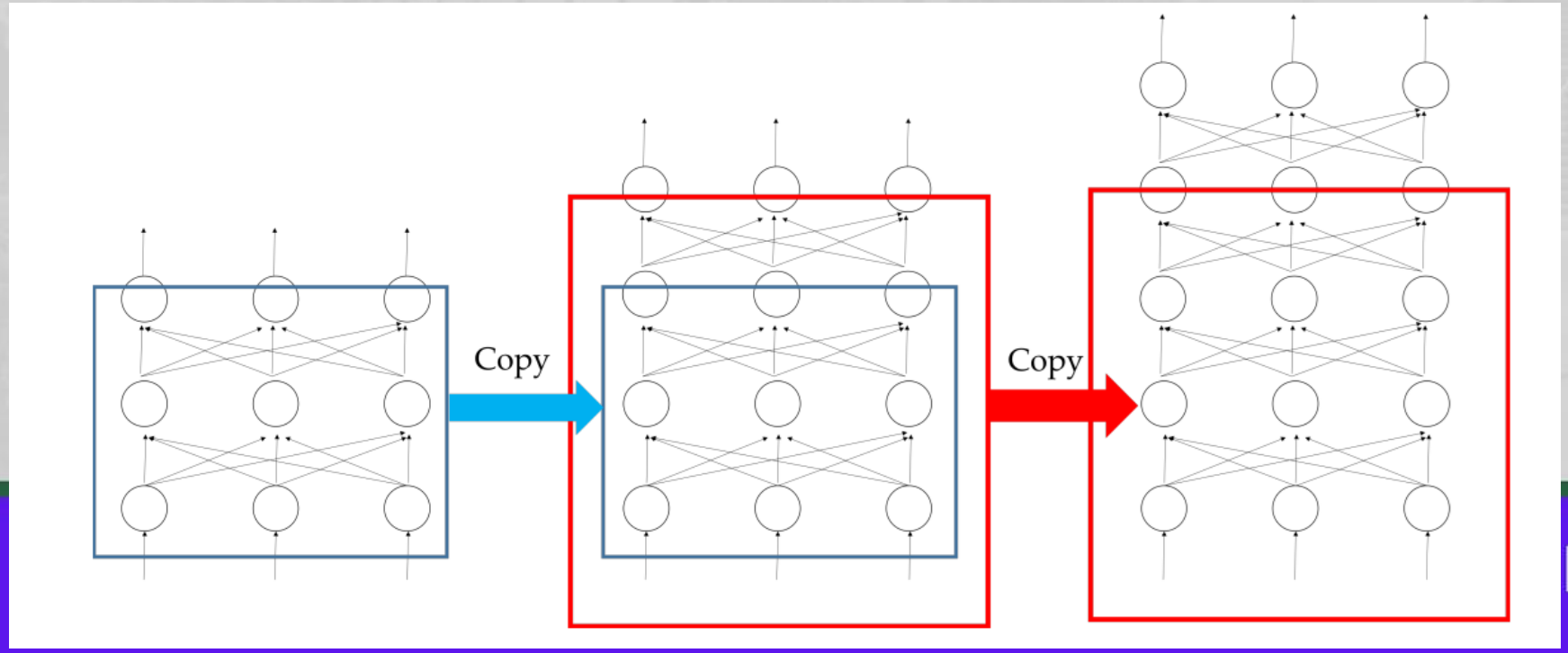
- Keuntungan utama deep learning adalah **mampu merubah data dari non-linearly separable menjadi linearly separable** melalui serangkaian **transformasi** (*hidden layers*).
- Selain itu, deep learning juga mampu mencari decision boundary yang berbentuk non-linear, serta mengsimulasikan interaksi non-linear antar fitur

Deep Neural Network

- Karena memiliki banyak parameter, proses latihan ANN pada umumnya lambat.
- Ada **beberapa strategi untuk mempercepat pembelajaran** menggunakan deep learning, misalnya: **regularisasi**, **successive learning**, dan **penggunaan autoencoder**.

Deep Neural Network

- Sebagai contoh, arti ***successive learning*** adalah jaringan yang dibangun secara bertahap.
- Misal kita latih ANN dengan 3 layers, kemudian kita lanjutkan 3 layers tersebut menjadi 4 layers, lalu kita latih lagi menjadi 5 layers, dst.



Minibatches

Minibatches

- Pada contoh yang diberikan, ***error atau loss*** dihitung per tiap data point.
- Artinya begitu ada melewatkan suatu input, parameter langsung dioptimisasi sesuai dengan ***loss***.
- Pada umumnya, hal ini tidak baik untuk dilakukan karena ANN menjadi tidak stabil.
- Metode yang lebih baik digunakan adalah teknik ***minibatches***. Yaitu **mengoptimisasi parameter untuk beberapa buah inputs**. Jadi, **update parameter dilakukan per batch**.

Minibatches

- Perhitungan error juga berubah, diberikan pada persamaan berikut dimana *B* melambangkan ***batch size*** (banyaknya ***instance per batch***), ***y*** adalah *desired output* dan ***o*** adalah *actual output*.

$$E(\text{minibatch}) = \frac{1}{B}\sum_{i=1}^{B}\|\mathbf{y}-\mathbf{o}\|^2$$

- Perhitungan *error* saat menggunakan *minibatches* secara sederhana adalah rata-rata (bisa diganti dengan penjumlahan saja) individual error untuk semua instance yang ada pada batch bersangkutan.
- Setelah menghitung *error per batch*, barulah *backpropagation* dilakukan

Minibatches

- Data mana saja yang dimasukkan ke suatu batch dalam dipilih secara acak.
- Seperti yang mungkin kamu sadari secara intuitif, urutan data yang disajikan saat training mempengaruhi kinerja ANN.
- Pengacakan ini menjadi penting agar ANN mampu men-generalisasi dengan baik.
- Kita dapat mengatur laju pembelajaran dengan menggunakan *learning rate*.

learning rate

- Pada library/API deep learning, *learning rate* pada umumnya berubah-ubah sesuai dengan waktu.
- Selain itu, tidak ada nilai khusus (*rule-of-thumb*) untuk *learning rate* terbaik. Pada umumnya, kita inisiasi *learning rate* dengan nilai {0.001, 0.01, 0.1}.
- Biasanya, kita **menginisiasi proses latihan dengan nilai learning rate cukup besar, kemudian mengecil seiring berjalannya waktu**.
- Kemudian, kita mencari konfigurasi parameter terbaik dengan metode ***grid-search***, yaitu dengan mencoba-coba parameter secara exhaustive (*brute-force*) kemudian memilih parameter yang memberikan kinerja terbaik

TERIMA KASIH