Pengantar Deep Learning untuk NLP

Alfan Farizki Wicaksono (alfan@cs.ui.ac.id)

Information Retrieval Lab.
Fakultas Ilmu Komputer
Universitas Indonesia
2017

Deep Learning Tsunami

"Deep Learning waves have lapped at the shores of computational linguistics for several years now, but 2015 seems like the year when the full force of the tsunami hit the major Natural Language Processing (NLP) conferences."

-Dr. Christopher D. Manning, Dec 2015

Christopher D. Manning. (2015). Computational Linguistics and Deep Learning Computational Linguistics, 41(4), 701–707.

Tips

Sebelum mempelajari RNNs dan arsitektur Deep Learning yang lainnya, disarankan untuk kita mempelajari beberapa topik berikut:

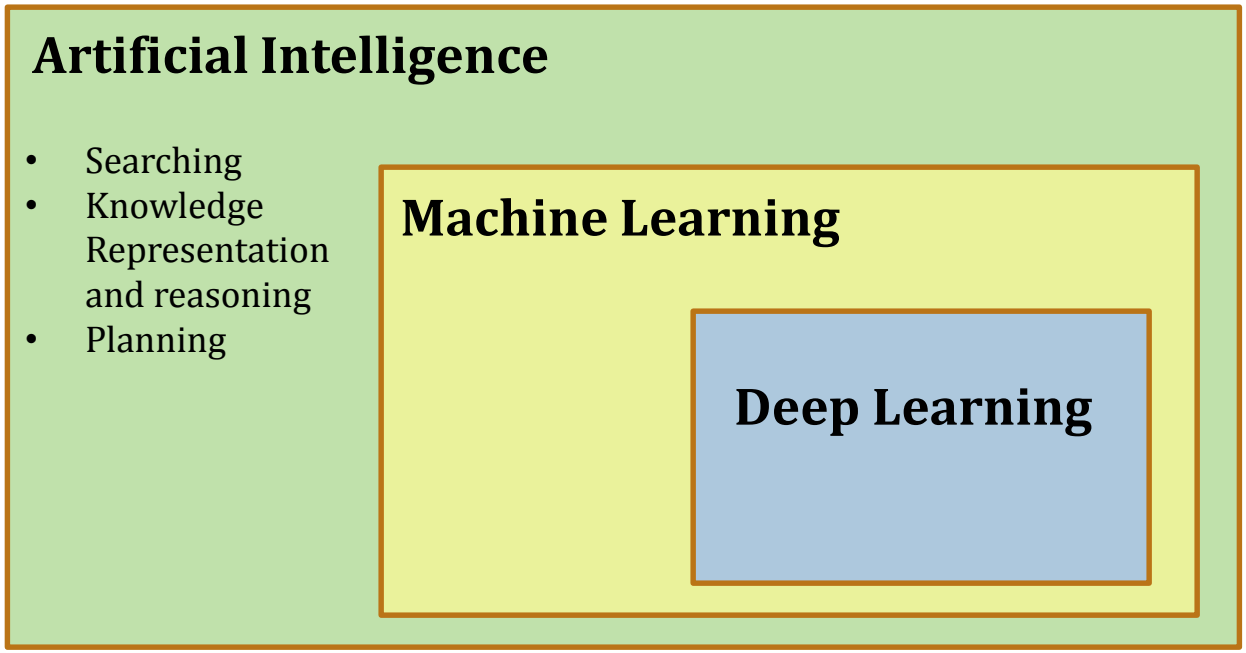
- Gradient Descent/Ascent
- Linear Regression
- Logistic Regression
- Konsep Backpropagation dan Computational Graph
- Multilayer Neural Networks

Referensi/Bacaan

- Andrej Karpathy' Blog
  - http://karpathy.github.io/2015/05/21/rnn-effectiveness/
- Colah's Blog
  - http://karpathy.github.io/2015/05/21/rnn-effectiveness/
- Buku Deep Learning Yoshua Bengio
  - Y. Bengio, Deep Learning, MLSS 2015

Deep Learning vs Machine Learning

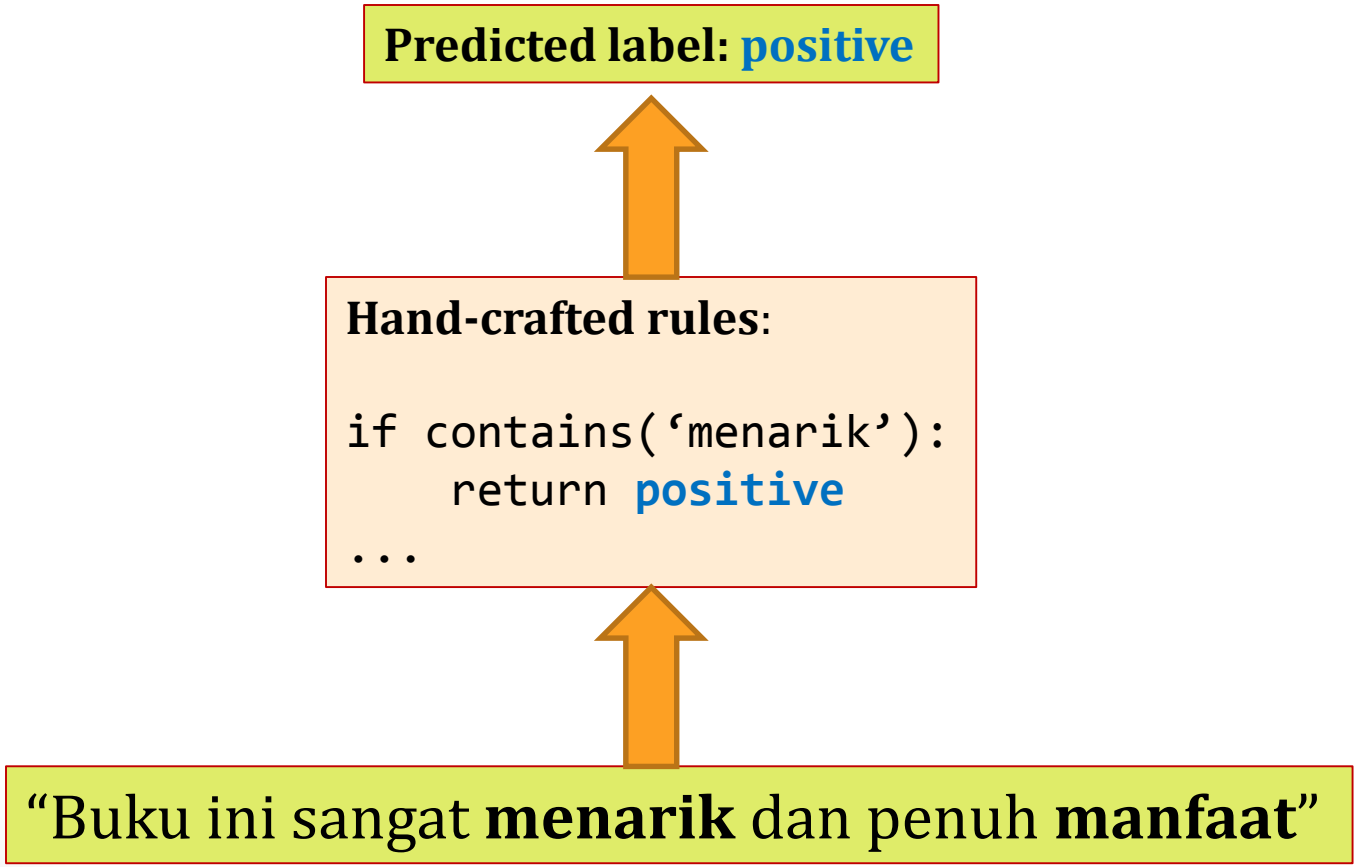
- Deep Learning adalah bagian dari isu Machine Learning
- Machine Learning adalah bagian dari isu Artificial Intelligence



Machine Learning
(rule-based)

: Dirancang manusia

: Di-infer otomatis

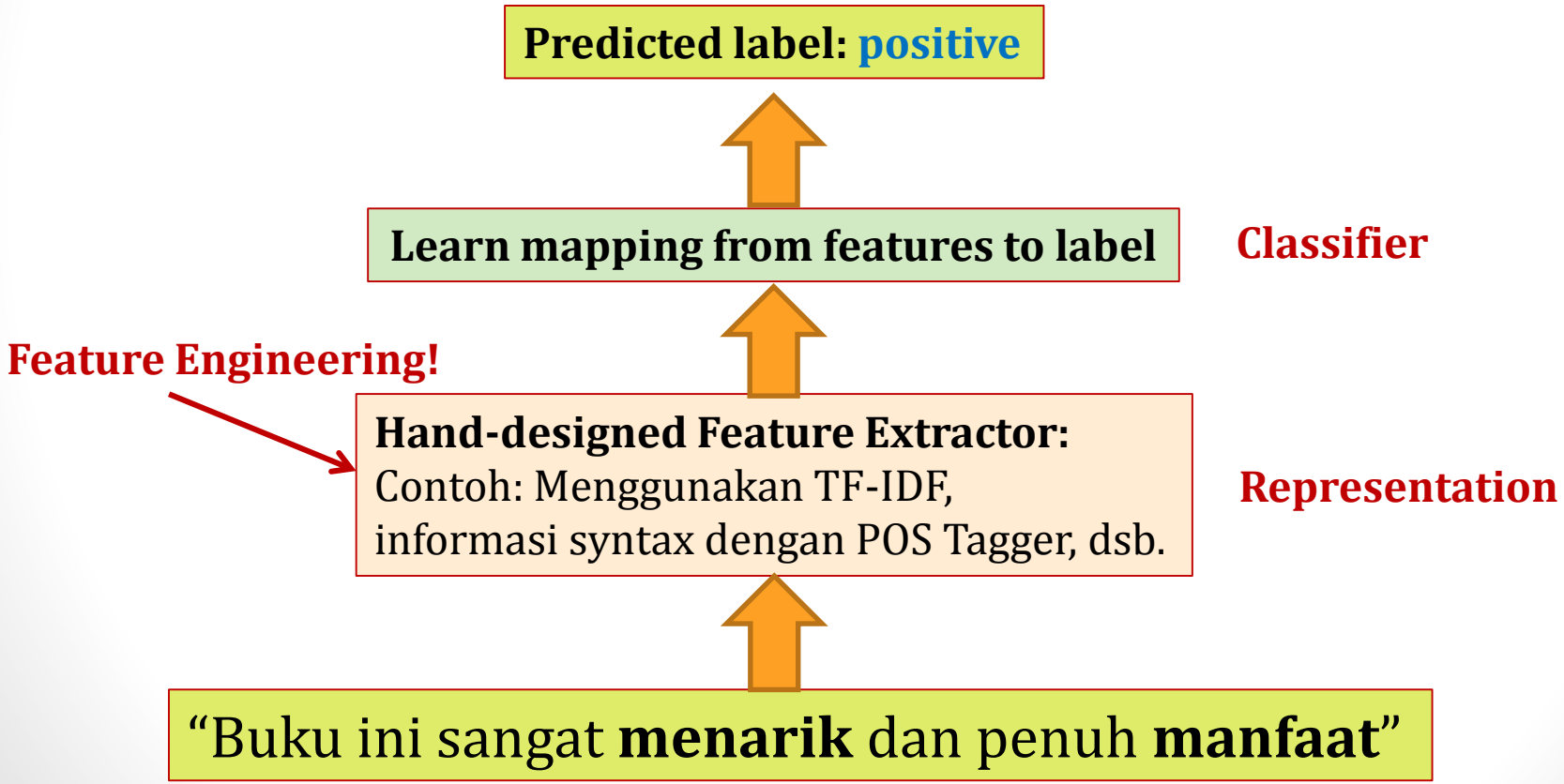


Machine Learning (classical ML)

: Dirancang manusia

: Di-infer otomatis

Fungsi klasifikasi dioptimasi berdasarkan input (fitur) & ouput.

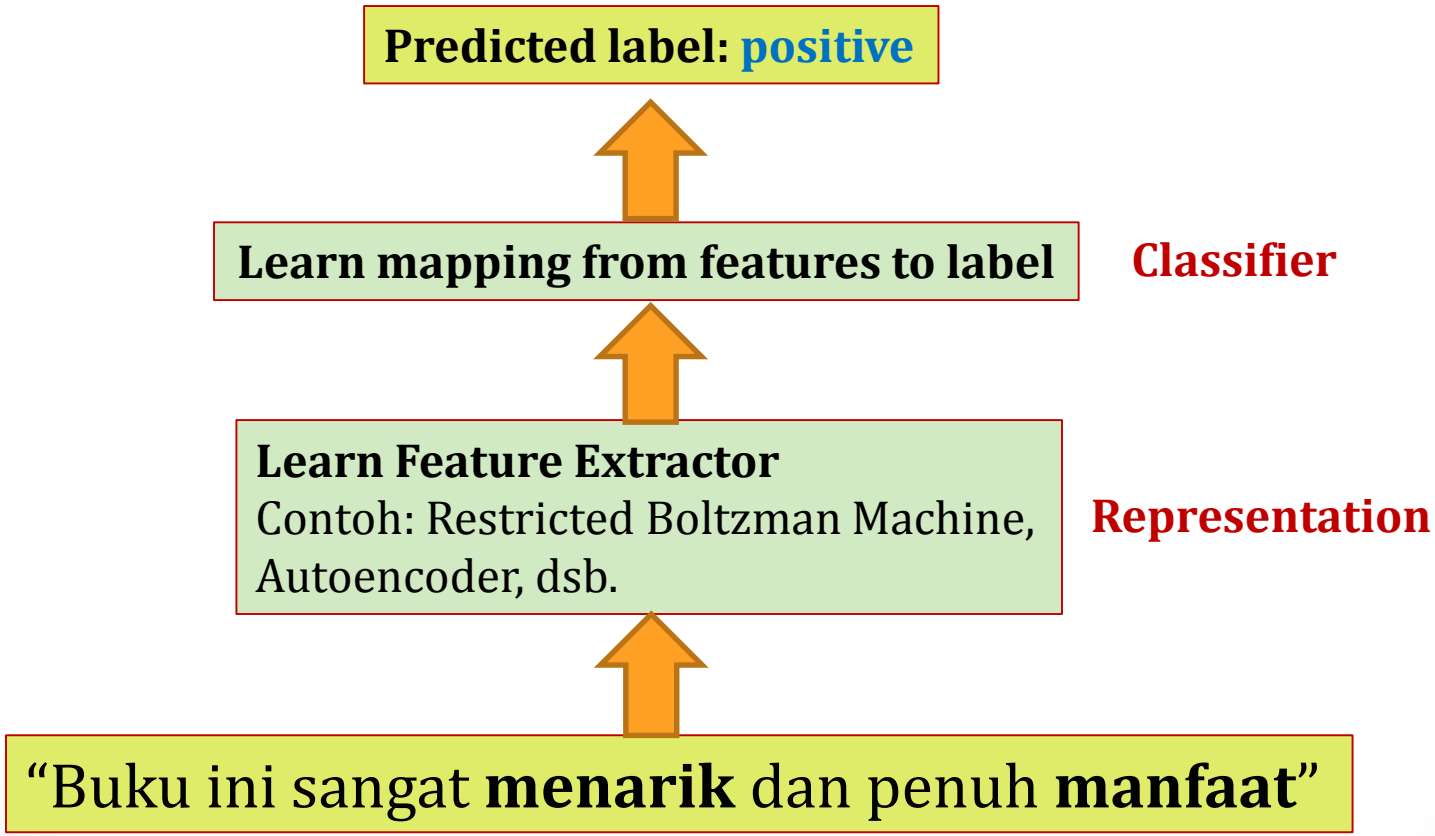


Machine Learning
(Representation Learning)

: Dirancang manusia

: Di-infer otomatis

Fitur dan fungsi klasifikasi di-optimasi secara bersama-sama.

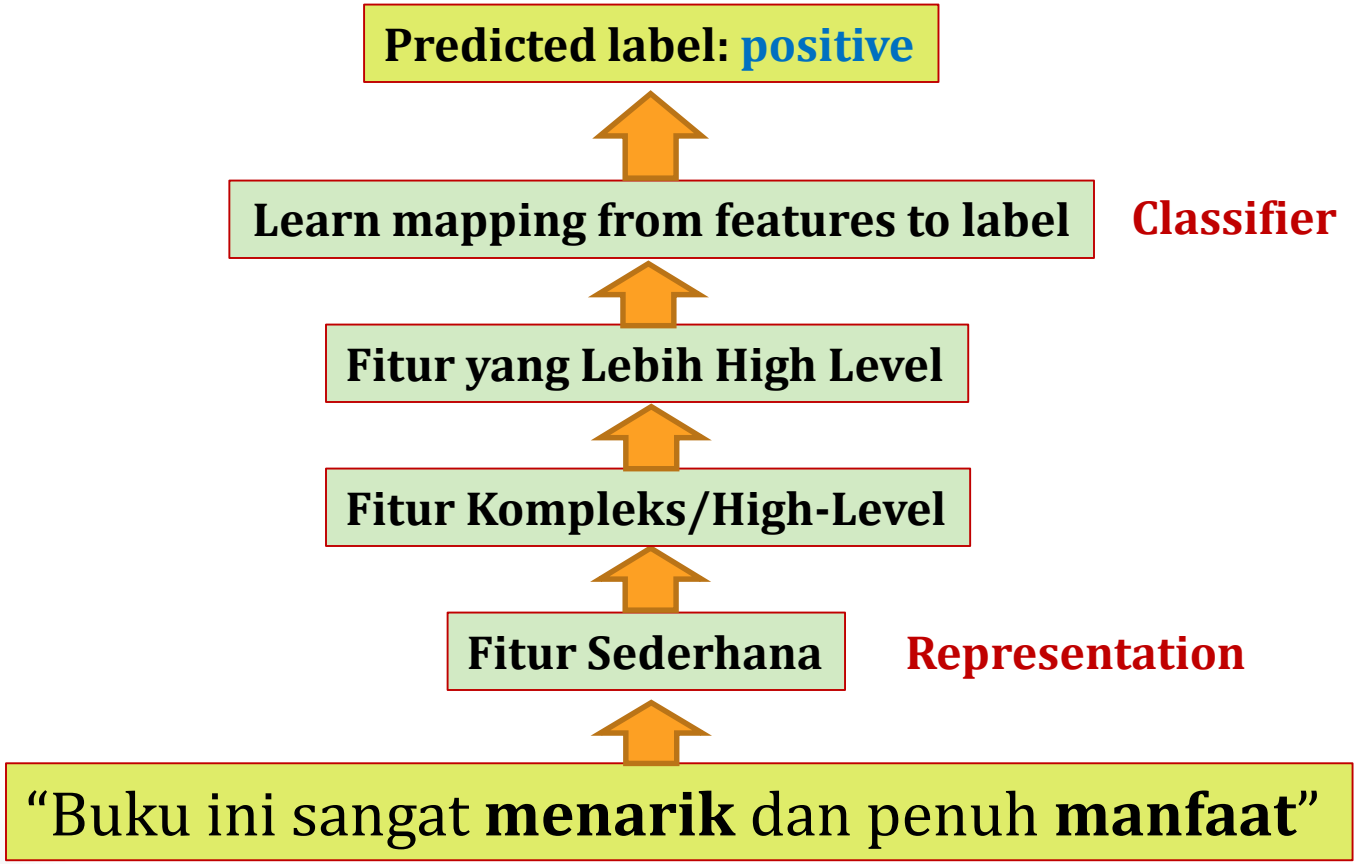


Machine Learning
(Deep Learning)

: Dirancang manusia

: Di-infer otomatis

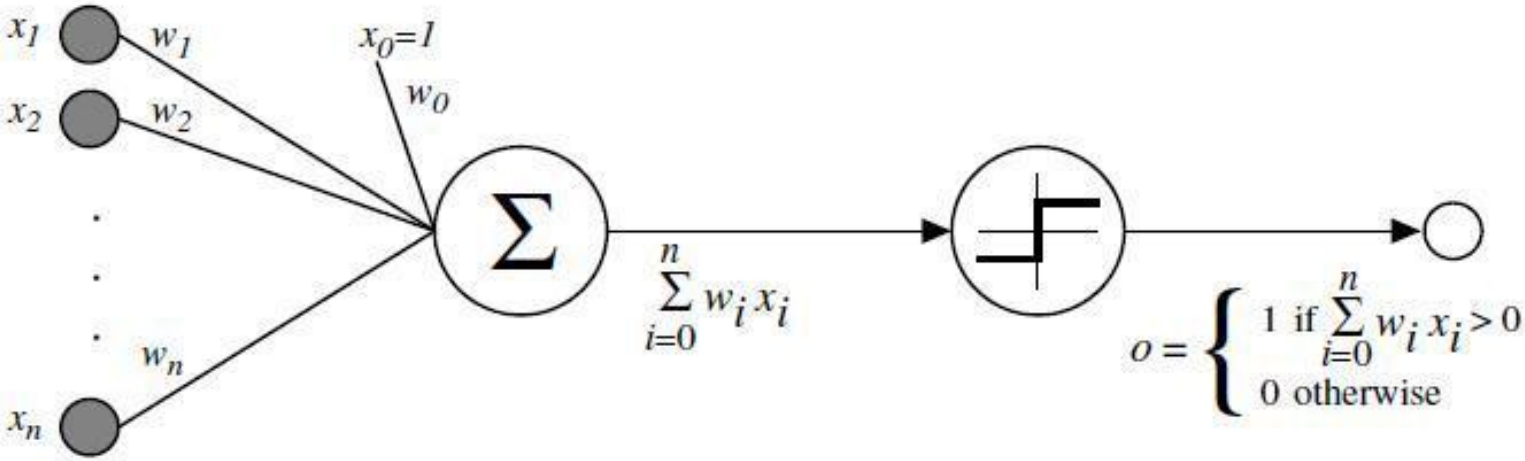
Deep Learning learns Features!



Sejarah

The Perceptron (Rosenblatt, 1958)

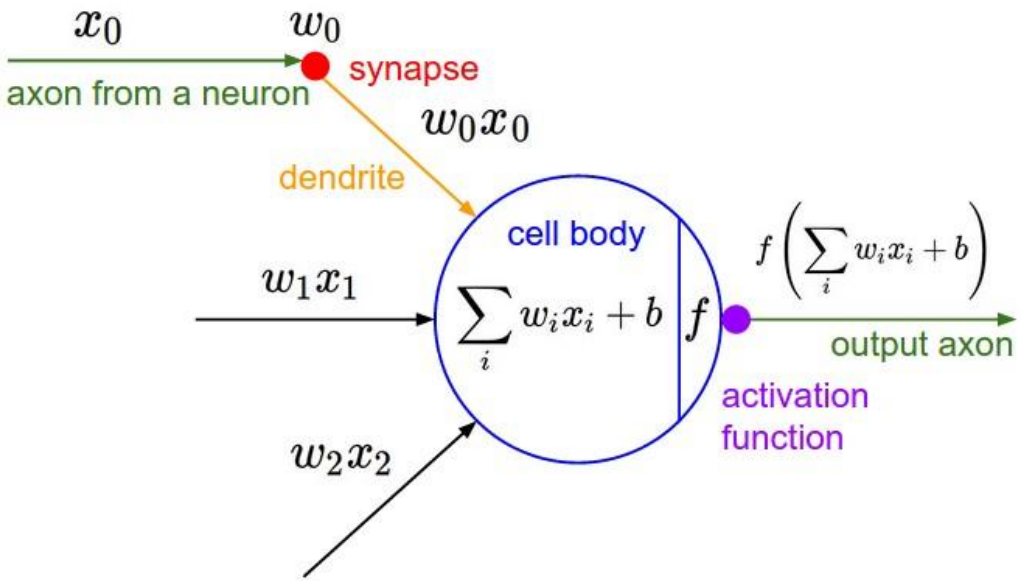
- Sejarah dimulai dimulai dengan **Perceptron** di akhir tahun 1950.
- Perceptron terdiri dari 3 layer: **Sensory**, **Association**, dan **Response**.



http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/

Rosenblatt, Frank. "The perceptron: a probabilistic model for information storage and organization in the brain." Psychological review 65.6 (1958): 386.

The Perceptron (Rosenblatt, 1958)



Activation function adalah fungsi non-linier. Dalam kasus perceptron Rosenblatt, activation function adalah operasi thresholding biasa (step function).

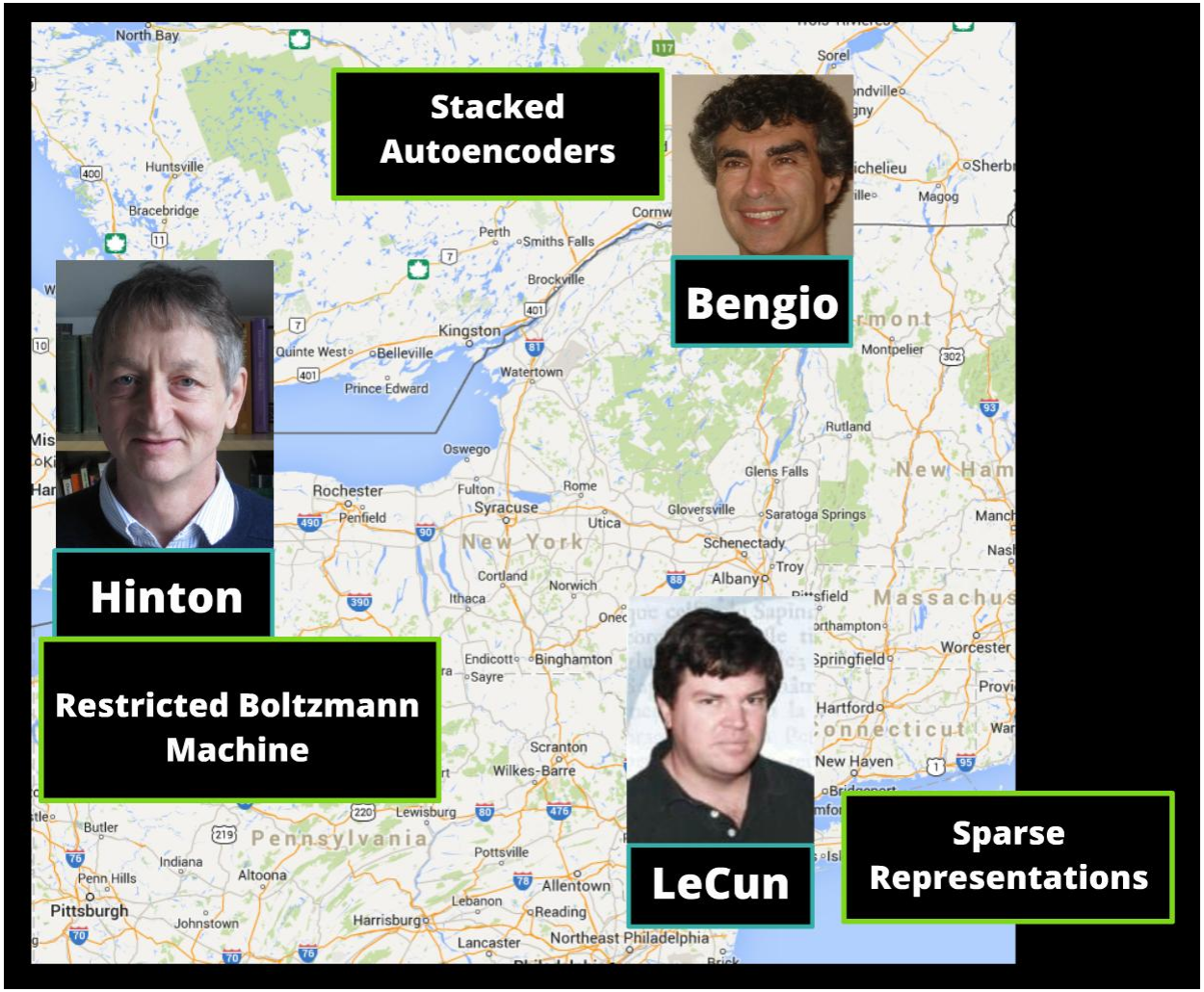
Learning perceptron menggunakan metode **Donald Hebb**.

Saat itu, mampu melakukan klasifikasi untuk input Pixel 20x20!

http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/

The organization of behavior: A neuropsychological theory. D. O. Hebb. John Wiley And Sons, Inc., New York, 1949

The Fathers of Deep Learning(?)



https://www.datarobot.com/blog/a-primer-on-deep-learning/

The Fathers of Deep Learning(?)

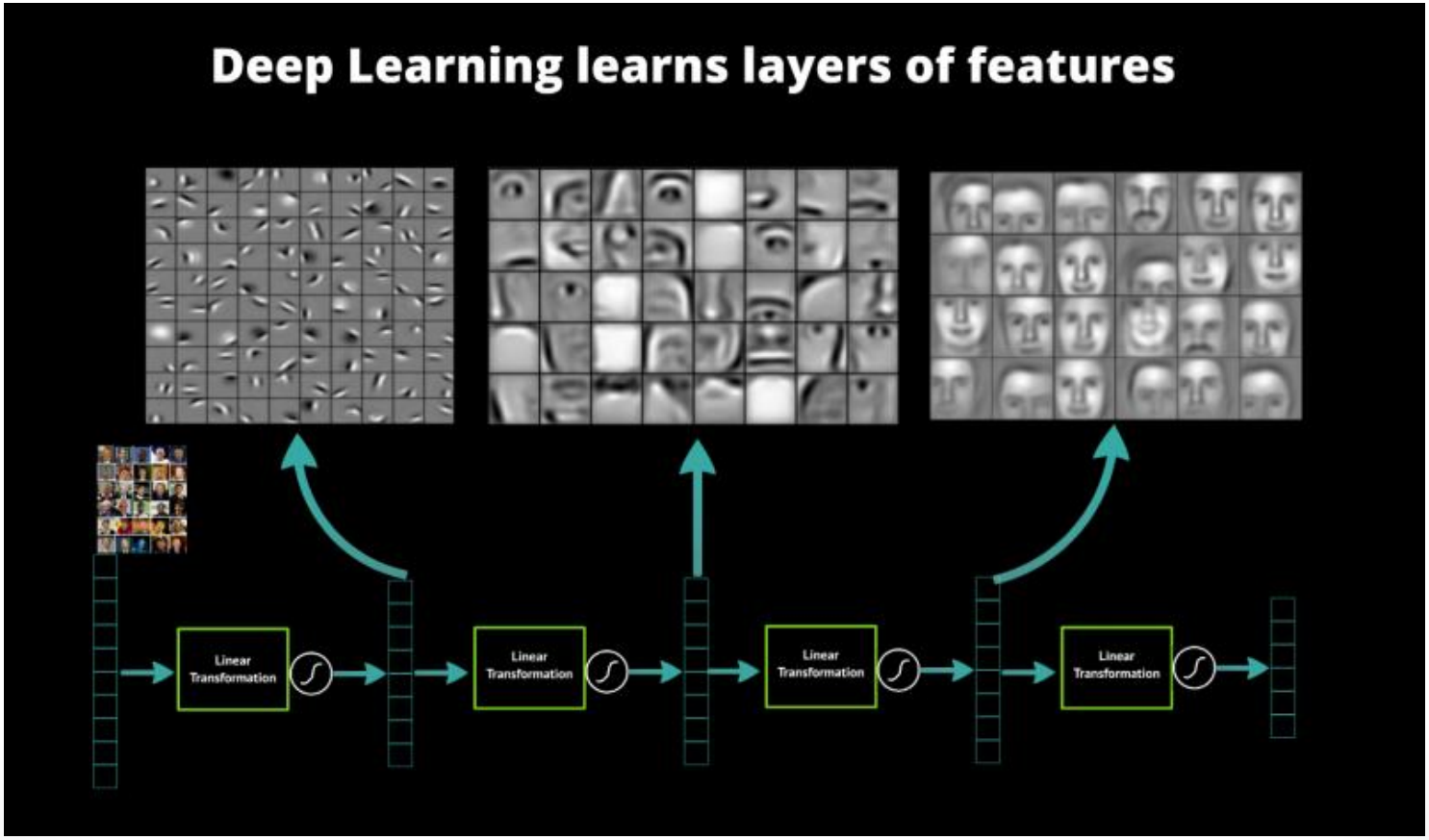
- Di tahun 2006, ketiga orang tersebut mengembangkan cara untuk memanfaatkan dan mengatasi masalah training terhadap deep neural networks.
- Sebelumnya, banyak orang yang sudah menyerah terkait manfaat dari neural network, dan cara training-nya.
- Mereka mengatasi masalah terkait Neural Network belum mampu **belajar untuk menemukan representasi yang berguna**.

- **Geoff Hinton** *has been snatched up by Google;*
- **Yann LeCun** *is Director of AI Research at Facebook;*
- **Yoshua Bengio** *holds a position as research chair for Artificial Intelligence at University of Montreal*

https://www.datarobot.com/blog/a-primer-on-deep-learning/

The Fathers of Deep Learning(?)

- *Automated learning of data representations and features is what the hype is all about!*



https://www.datarobot.com/blog/a-primer-on-deep-learning/

Mengapa sebelumnya “deep learning” tidak sukses?

- Sebenarnya, neural network kompleks sudah banyak ditemukan sebelumnya.
- Bahkan **Long-Short Term Memory (LSTM)** network, yang saat ini ramai digunakan di bidang NLP, ditemukan tahun **1997 oleh Hochreiter & Schmidhuber**.

- Ditambah lagi, orang dahulu percaya bahwa neural network “**can solve everything!**”. Tetapi, mengapa mereka tidak bisa melakukannya dahulu?

Mengapa sebelumnya "deep learning" tidak sukses?

- Beberapa alasan, oleh **Ilya Sutskever**:
  - http://yyue.blogspot.co.id/2015/01/a-brief-overview-of-deep-learning.html

- **Computers were slow.** *So the neural networks of past were tiny. And tiny neural networks cannot achieve very high performance on anything. In other words, small neural networks are not powerful*.
- **Datasets were small.** *So even if it was somehow magically possible to train LDNNs, there were no large datasets that had enough information to constrain their numerous parameters. So failure was inevitable*.
- **Nobody knew how to train deep nets.** *The current best object recognition networks have between 20 and 25 successive layers of convolutions. A 2 layer neural network cannot do anything good on object recognition. Yet back in the day everyone was very sure that* ***deep nets*** *cannot be trained with* ***SGD****, since that would've been too good to be true*

The Success of Deep Learning

Salah satu faktor-nya adalah karena saat ini ditemukan cara learning yang bekerja secara praktikal.

> *The success of Deep Learning hinges on a very fortunate fact: that well-tuned and carefully-initialized stochastic gradient descent (SGD) can train LDNNs on problems that occur in practice. It is not a trivial fact since the training error of a neural network as a function of its weights is highly non-convex. And when it comes to* ***non-convex optimization****, we were taught that all* ***bets are off****...*
>
> *And yet, somehow, SGD seems to be very good at training those large deep neural networks on the tasks that we care about. The problem of training neural networks is* ***NP-hard****, and in fact there exists a family of datasets such that the problem of finding the best neural network with three hidden units is* ***NP-hard****. And yet,* ***SGD*** *just solves it in practice.*

Ilya Sutskever, http://yyue.blogspot.co.at/2015/01/a-brief-overview-of-deep-learning.html

Apa Itu Deep Learning?

Apa itu Deep Learning?

- Kenyataannya, **Deep Learning = (Deep) Artificial Neural Networks (ANNs)**
- Dan **Neural Networks** sebenarnya adalah sebuah **Tumpukan Fungsi Matematika**

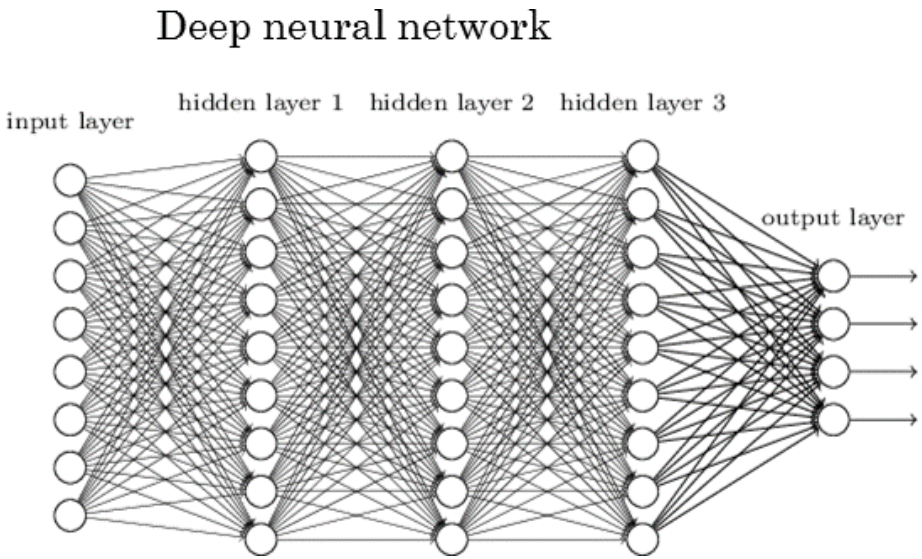
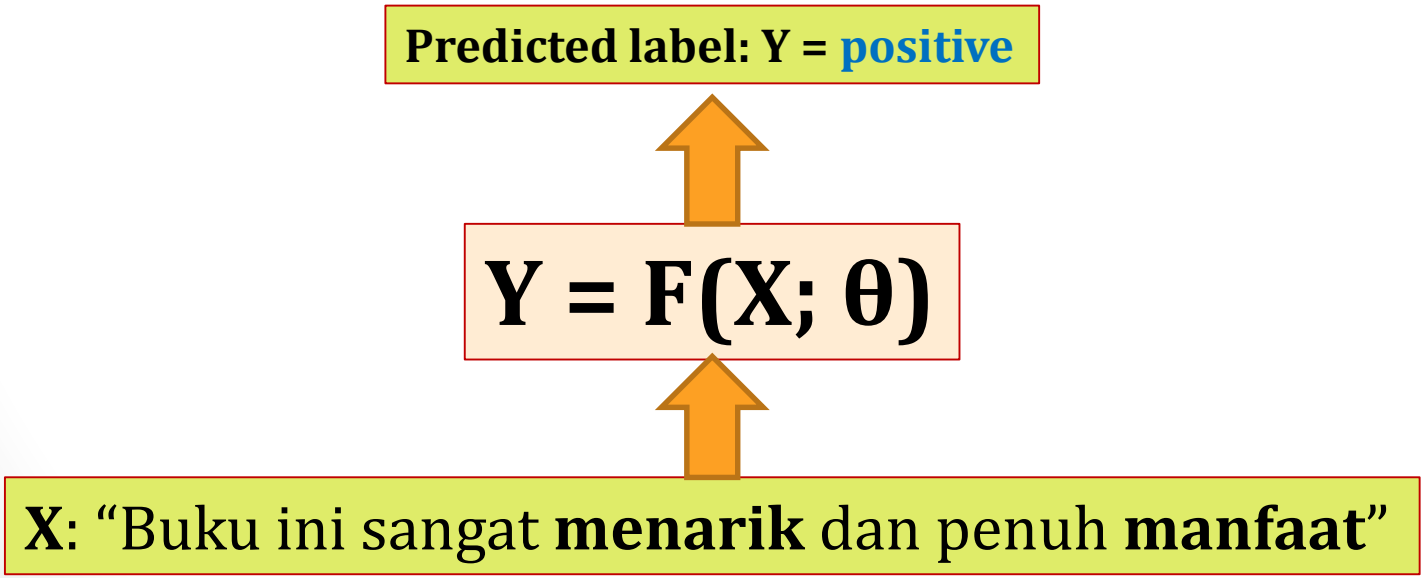


Image Courtesy: Google

Apa itu Deep Learning?

Secara praktis, **(supervised) Machine Learning** itu adalah: Ekspresikan permasalahan ke dalam sebuah fungsi **F** (yang mempunyai parameter **θ**), lalu secara otomatis cari parameter **θ** sehingga fungsi **F** tepat mengeluarkan output yang diinginkan.

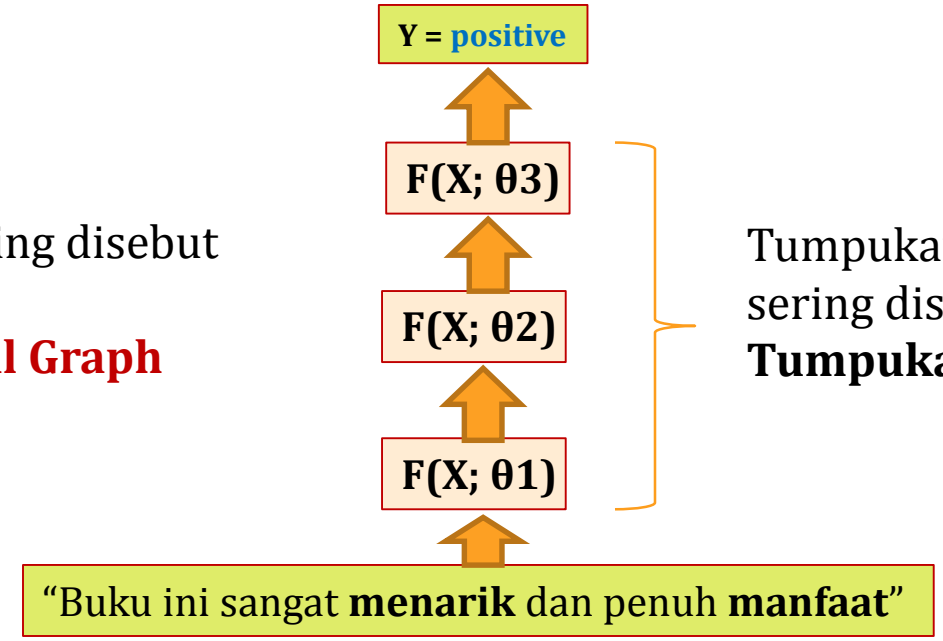


Apa itu Deep Learning?

Untuk **Deep Learning**, fungsi tersebut biasanya terdiri dari **tumpukan banyak fungsi** yang biasanya serupa.

$$Y = F(F(F(X;\theta_3);\theta_2);\theta_1)$$

Gambar ini sering disebut dengan istilah **Computational Graph**



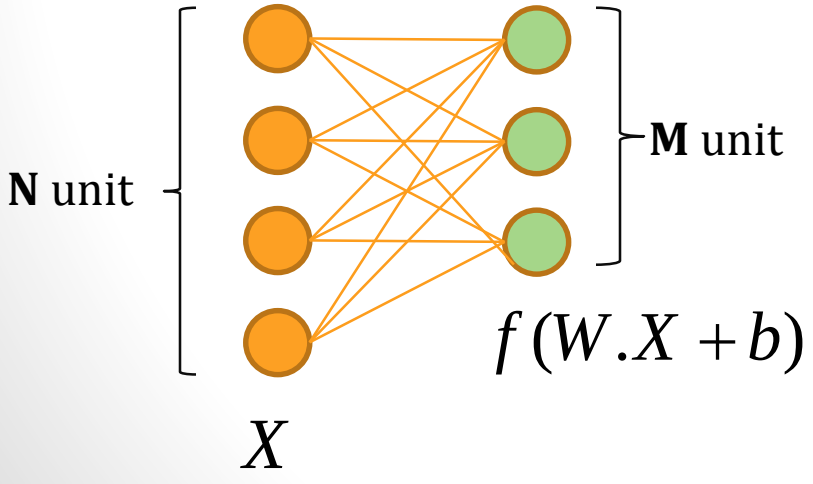
Tumpukan Fungsi ini sering disebut dengan **Tumpukan Layer**

Apa itu Deep Learning?

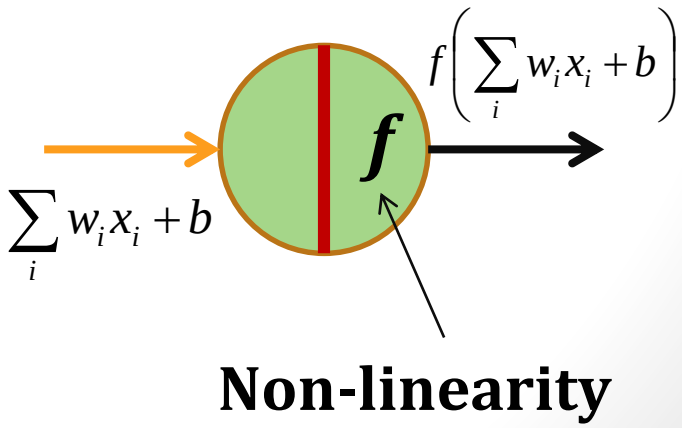
- **Layer** yang paling terkenal/umum adalah **Fully-Connected Layer**.

$$Y = F(X) = f(W.X + b)$$

- "*weighted sum of its inputs, followed by a **non-linear function***"
- Fungsi non-linier yang umum digunakan: **Tanh** (tangent hyperbolic), **Sigmoid**, **ReLU** (Rectified Linear Unit)



$$W \in R^{M \times N}$$
$$X \in R^{N}$$
$$b \in R^{M}$$



Mengapa perlu “Deep”?

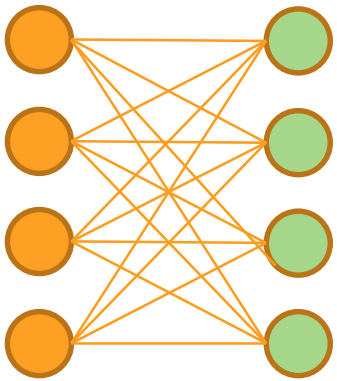
- Humans organize their ideas and concepts hierarchically
- Humans first learn simpler concepts and then compose them to represent more abstract ones
- Engineers break-up solutions into multiple levels of abstraction and processing
- It would be good to automatically learn / discover these concepts

Y. Bengio, Deep Learning, MLSS 2015, Austin, Texas, Jan 2014

(Bengio & Delalleau 2011)

Neural Networks

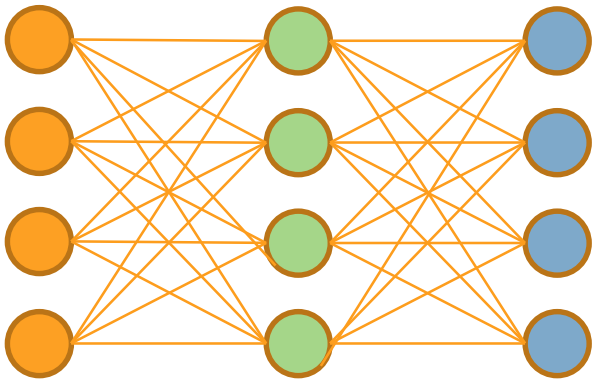
$$Y = f(W_1.X + b_1)$$



X $\quad$ $Y = f(W_1.X + b_1)$

Neural Networks

$$Y = f(W_1.(f(W_1.X + b_2)) + b_1)$$

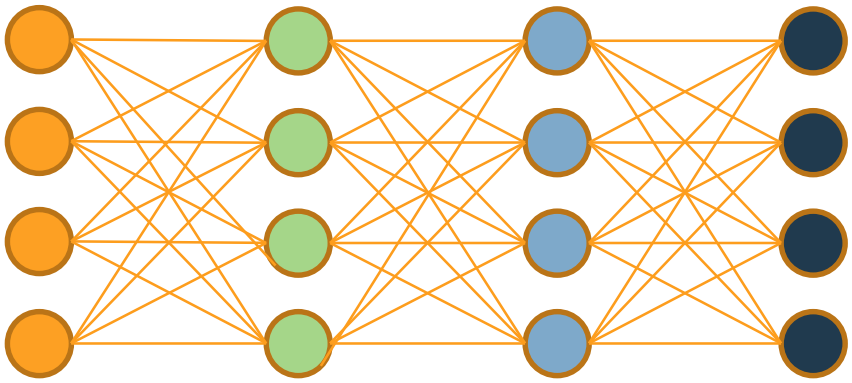


$$Y = f(W_2.H_1 + b_2)$$

X $\quad$ $H_1 = f(W_1.X + b_1)$

Neural Networks

$$Y = f(W_1.(f(W_2.(f(W_3.X + b_3)) + b_2)) + b_1)$$



$$Y = f(W_3.H_2 + b_3)$$

X

$$H_1 = f(W_1.X + b_1) \quad H_2 = f(W_2.H_1 + b_2)$$

Alasan matematis mengapa harus “deep”?

- A neural network with a **single hidden layer** of *enough units* can approximate **any continuous function** arbitrarily well.

- In other words, it can solve whatever problem you’re interested in!

(Cybenko 1998, Hornik 1991)

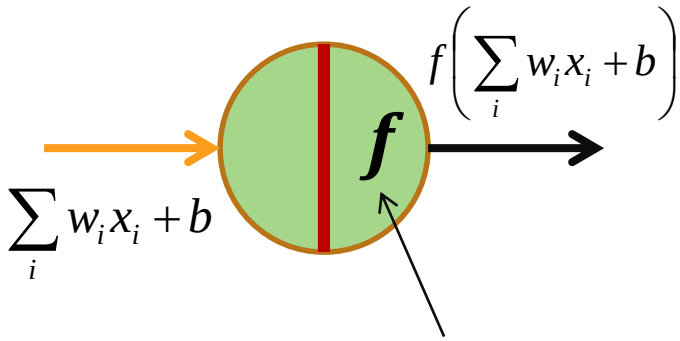
Alasan matematis mengapa harus “deep”?

Akan tetapi ...

- “**Enough units**” can be a very large number. There are functions representable with a **small, but deep** network that would require **exponentially** many units with a single layer.
- The proof only says that a shallow network *exists*, it does not say how to find it.
- Evidence indicates that it is **easier to train a deep network to perform well than a shallow one.**
- A more recent result brings an example of a very large class of functions that cannot be efficiently represented with a **small-depth** network.

(e.g., Hastad et al. 1986, Bengio & Delalleau 2011)
(Braverman, 2011)

Mengapa perlu *non-linearity*?



Mengapa perlu fungsi non-linier ***f***?

$$H_1 = W_1.X + b_1$$
$$H_2 = W_2.H_1 + b_2$$
$$Y = W_3.H_2 + b_3$$

?

Kalau tanpa ***f***, rangkaian fungsi ini adalah **tetap fungsi linier**.

Data bisa sangat kompleks, dan terkadang hubungan yang ada pada data **tidak hanya linier, tetapi bisa non-linier**. Perlu representasi yang bisa menangkap hal ini.

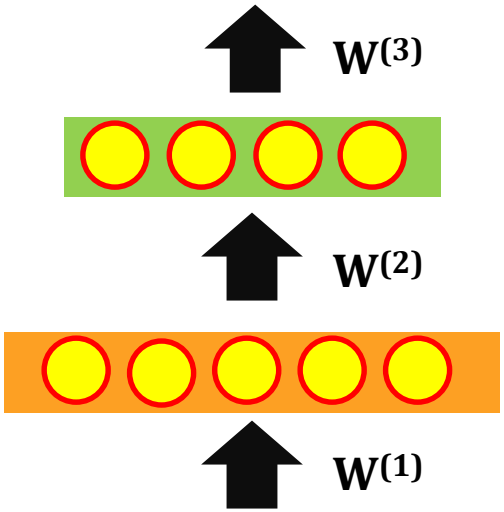
Training Neural Networks

$$Y = f(W_1.(f(W_2.(f(W_3.X + b_3)) + b_2)) + b_1)$$

- Secara **random**, kita inisialisasi semua parameter $\mathbf{W_1}$, $\mathbf{b_1}$, $\mathbf{W_2}$, $\mathbf{b_2}$, $\mathbf{W_3}$, $\mathbf{b_3}$
- Definisikan sebuah **cost function/loss function** yang mengukur seberapa baik fungsi neural network Anda.
  - Seberapa jauh nilai yang diprediksi dengan nilai sesungguhnya
- Secara iteratif/berulang-ulang, sesuaikan nilai parameter sehingga nilai **loss function** menjadi **minimal**.

Training Neural Networks

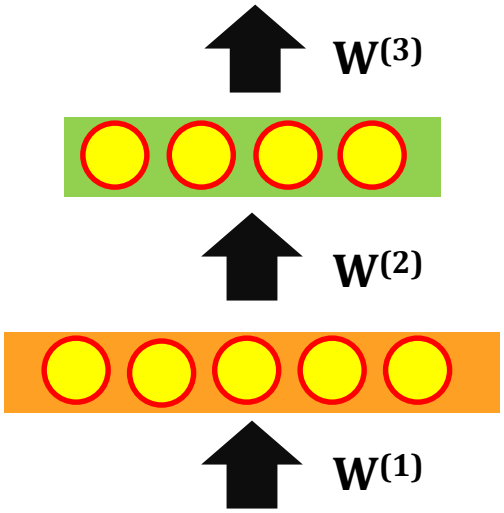
- Initialize trainable parameters randomly



Buku

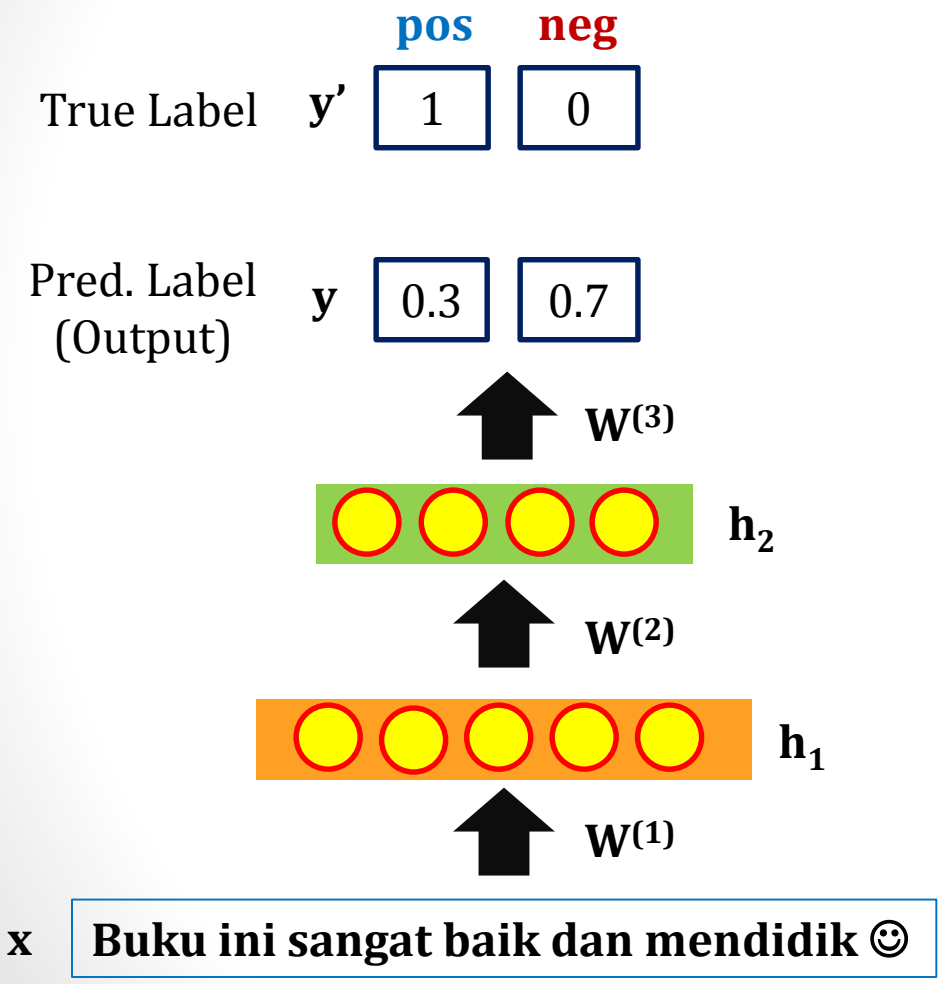
Training Neural Networks

- Initialize trainable parameters randomly
- **Loop: x = 1 → #epoch**:
  - Pick a training example



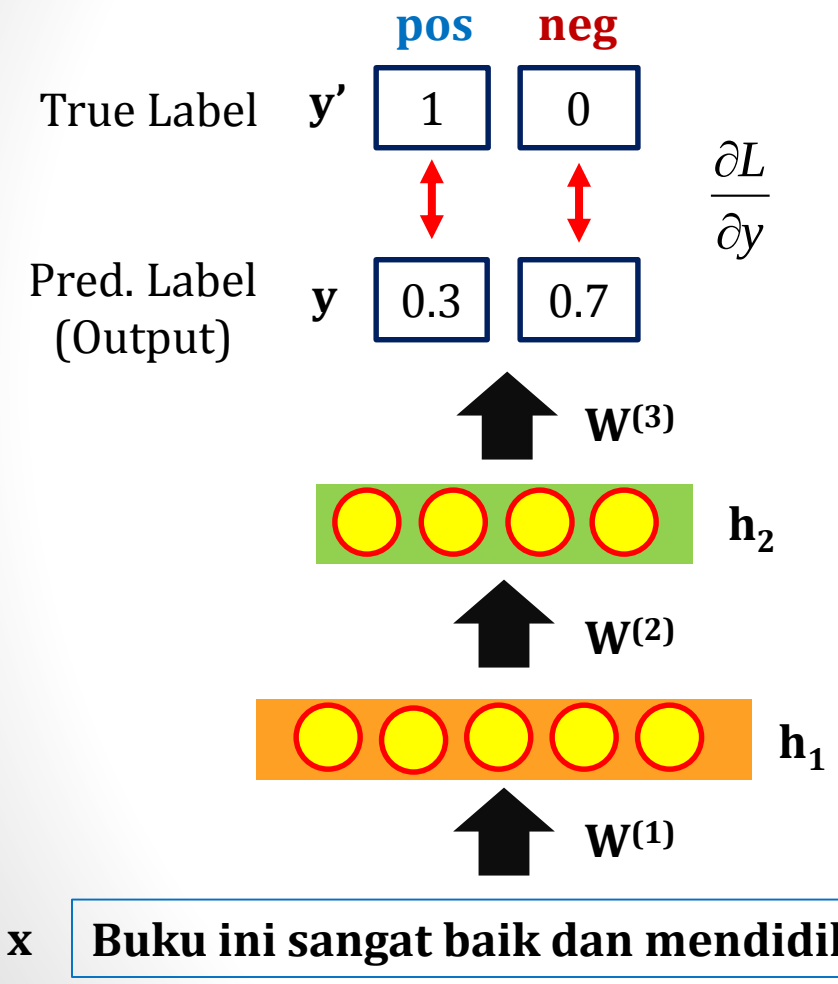
x **Buku ini sangat baik dan mendidik ☺**

Training Neural Networks



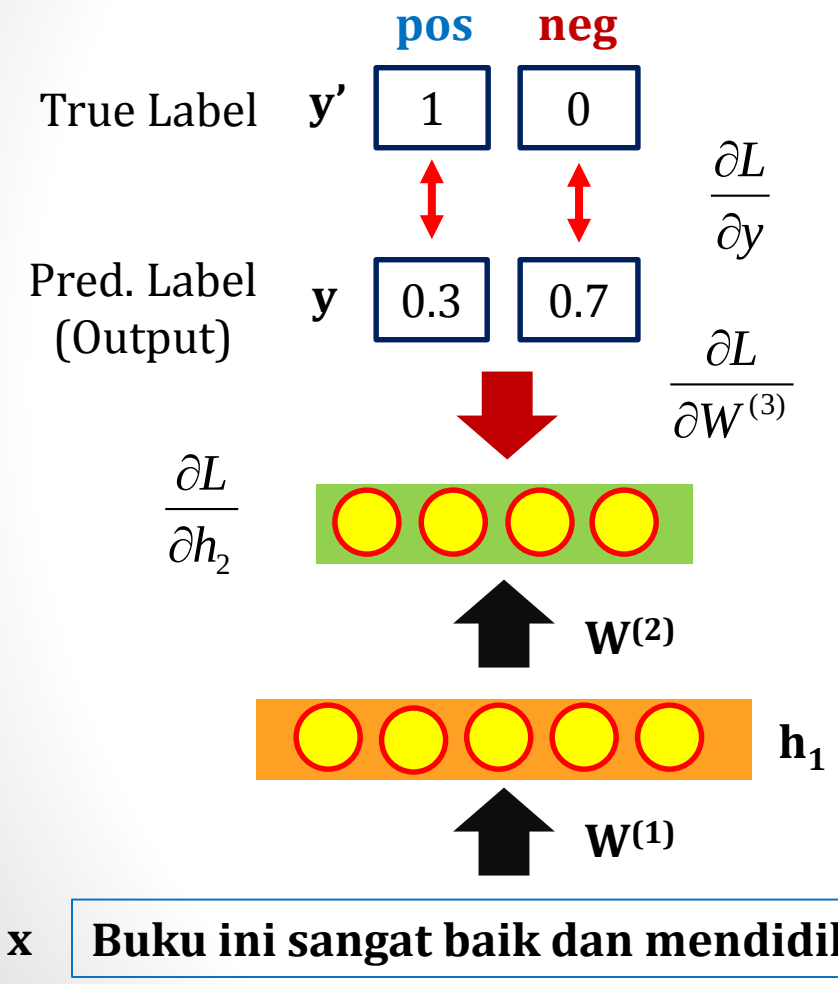
- Initialize trainable parameters randomly
- **Loop: x = 1 → #epoch**:
  - Pick a training example
  - Compute output by doing feed-forward process

Training Neural Networks



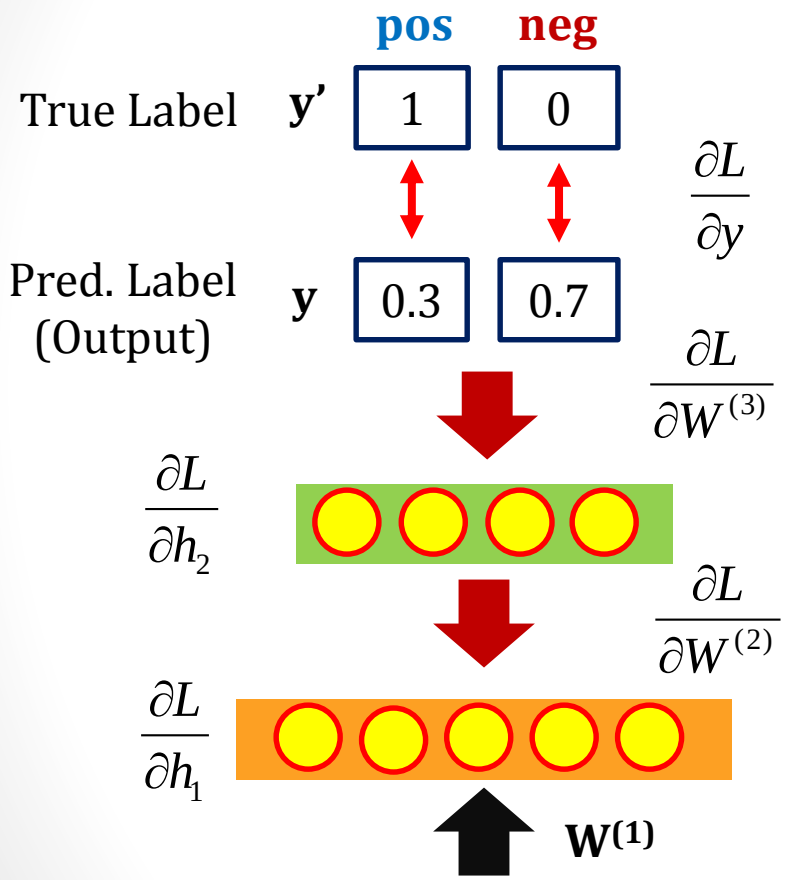
- Initialize trainable parameters randomly
- **Loop: x = 1 → #epoch**:
  - Pick a training example
  - Compute output by doing feed-forward process
  - Compute **gradient of loss** w.r.t. output

Training Neural Networks



- Initialize trainable parameters randomly
- **Loop: x = 1 → #epoch**:
  - Pick a training example
  - Compute output by doing feed-forward process
  - Compute **gradient of loss** w.r.t. output
  - Backpropagate loss, computing gradients w.r.t **trainable parameters**. It's like computing **contribution of error to the output** of each parameter

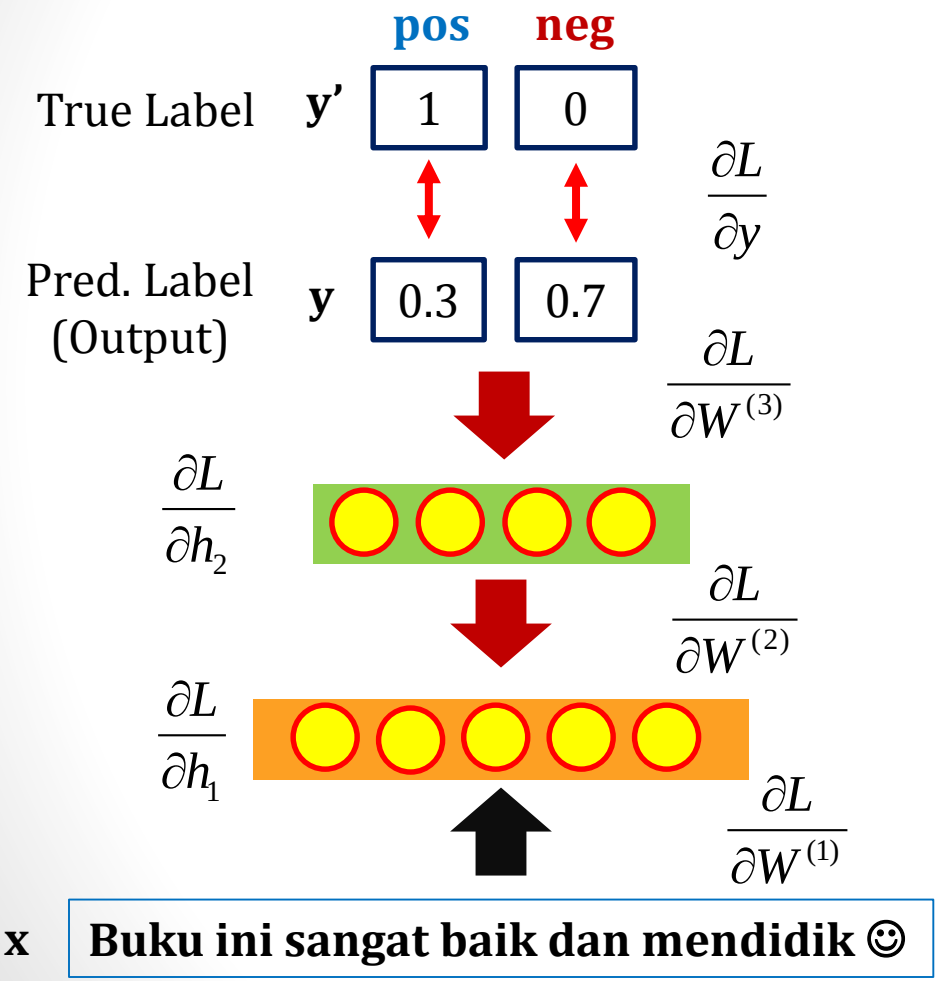
Training Neural Networks



x **Buku ini sangat baik dan mendidik ☺**

- Initialize trainable parameters randomly
- **Loop: x = 1 → #epoch**:
  - Pick a training example
  - Compute output by doing feed-forward process
  - Compute **gradient of loss** w.r.t. output
  - Backpropagate loss, computing gradients w.r.t **trainable parameters**. It's like computing **contribution of error to the output** of each parameter

Training Neural Networks



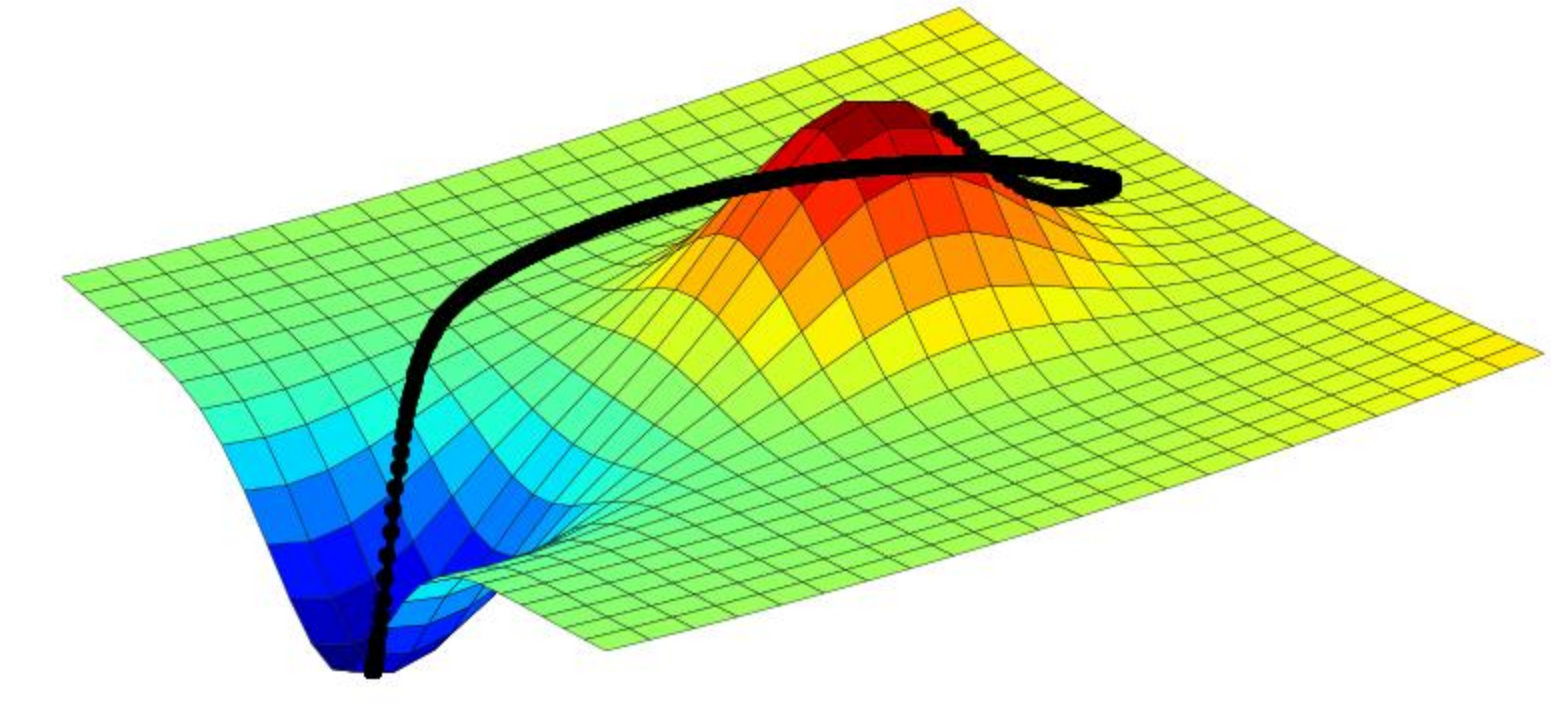
x | **Buku ini sangat baik dan mendidik ☺**

- Initialize trainable parameters randomly
- **Loop: x = 1 → #epoch**:
  - Pick a training example
  - Compute output by doing feed-forward process
  - Compute **gradient of loss** w.r.t. output
  - Backpropagate loss, computing gradients w.r.t **trainable parameters**. It's like computing **contribution of error to the output** of each parameter

Gradient Descent (GD)

Take small step in direction of negative gradient!

$$\theta_n^{(t+1)} \leftarrow \theta_n^{(t)} - \alpha_t \frac{\partial}{\partial \theta_n^{(t)}} f(\theta^{(t)})$$



https://github.com/joshdk/pygradesc

Lebih Detail dalam Hal Teknis …

Gradient Descent (GD)

Salah satu framework paling sederhana untuk permasalahan optimasi (*multivariate optimization*).

Digunakan untuk mencari konfigurasi parameter-parameter sehingga *cost function* menjadi optimal, dalam hal ini mencapai ***local minimum***.

GD menggunakan metode **iteratif**, yang dimulai dari sebuah titik acak, dan secara perlahan-lahan mengikuti arah negatif dari gradient sehingga pada akhirnya akan berpindah ke suatu titik kritikal, yang diharapkan merupakan ***local minimum***.

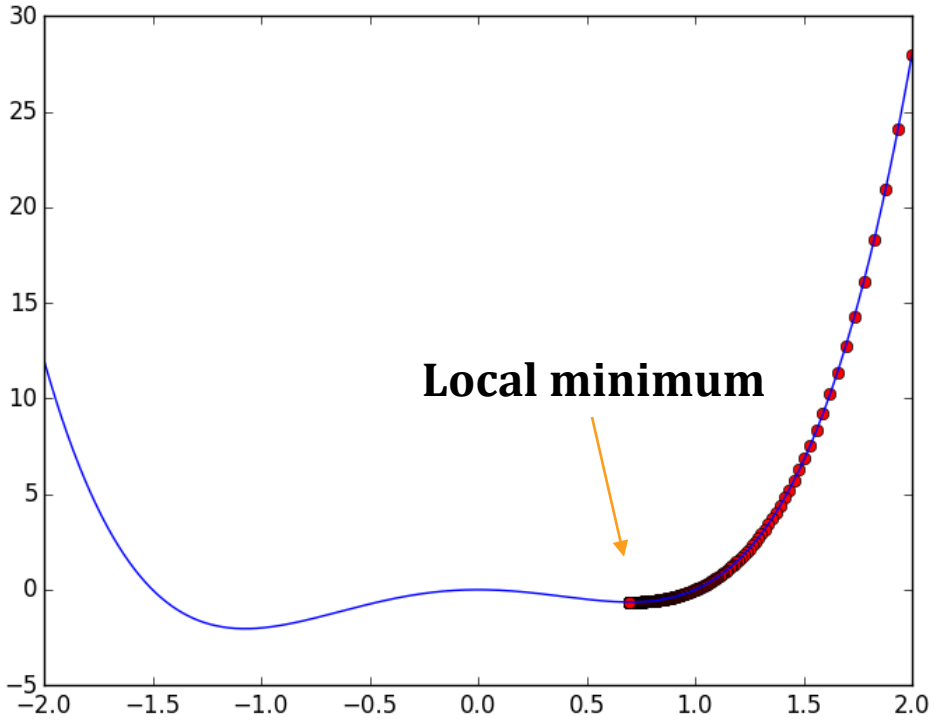
Tidak dijamin mencapai ***global minimum*** !

Gradient Descent (GD)

Problem: carilah nilai **x** sehingga fungsi $f(x) = 2x^4 + x^3 - 3x^2$ mencapai titik ***local minimum***.

Misal, kita pilih **x** dimulai dari **x=2.0**:

Algoritma GD konvergen pada titik **x = 0.699**, yang merupakan local minimum.



Gradient Descent (GD)

Algorithm:

$$For\ t = 1, 2, \ldots, N_{\max}:$$

$$x_{t+1} \leftarrow x_t - \alpha_t f'(x_t)$$

$$If\ \left\| f'(x_{t+1}) \right\| < \varepsilon\ then\ return\ "converged\ on\ critical\ point"$$

$$If\ \left\| x_t - x_{t+1} \right\| < \varepsilon\ then\ return\ "converged\ on\ x\ value"$$

$$If\ \ f(x_{t+1}) > f(x_t)\ then\ return\ "diverging"$$

$\boldsymbol{\alpha_t}$: ***learning rate*** atau ***step size*** pada iterasi ke-t
$\boldsymbol{\epsilon}$: sebuah bilangan yang sangat kecil
$\mathbf{N_{max}}$: batas banyaknya iterasi, atau disebut **epoch** jika iterasi selalu sampai akhir

Algoritma dimulai **dengan menebak nilai x_1** !

Tips: pilih $\boldsymbol{\alpha_t}$ yang tidak terlalu kecil, juga tidak terlalu besar.

Gradient Descent (GD)

Kalau parameter-nya ada banyak ?

Carilah $\theta = \theta_1, \theta_2, \ldots, \theta_n$ sehingga $f(\theta_1, \theta_2, \ldots, \theta_n)$ mencapai local minimum !

while not converged :

$$\theta_1^{(t+1)} \leftarrow \theta_1^{(t)} - \alpha_t \frac{\partial}{\partial \theta_1^{(t)}} f(\theta^{(t)})$$

$$\theta_2^{(t+1)} \leftarrow \theta_2^{(t)} - \alpha_t \frac{\partial}{\partial \theta_2^{(t)}} f(\theta^{(t)})$$

$\ldots$

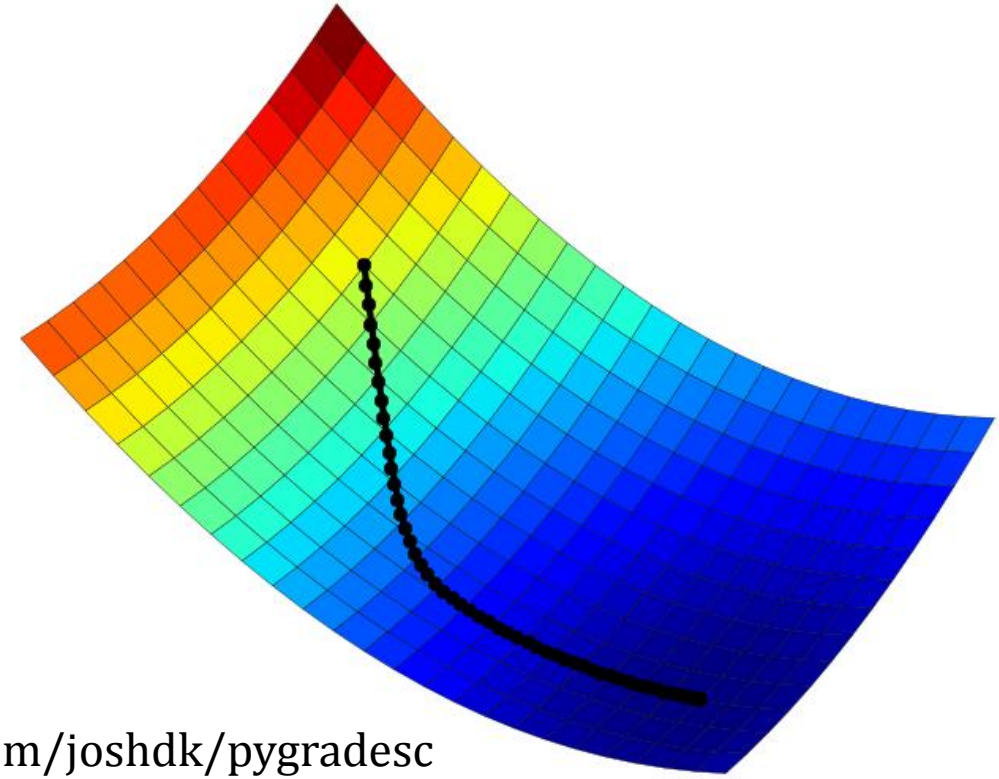
$$\theta_n^{(t+1)} \leftarrow \theta_n^{(t)} - \alpha_t \frac{\partial}{\partial \theta_n^{(t)}} f(\theta^{(t)})$$

Dimulai dengan menebak nilai awal **$\theta = \theta_1, \theta_2, \ldots, \theta_n$**

Gradient Descent (GD)

Take small step in direction of negative gradient!

$$\theta_n^{(t+1)} \leftarrow \theta_n^{(t)} - \alpha_t \frac{\partial}{\partial \theta_n^{(t)}} f(\theta^{(t)})$$



https://github.com/joshdk/pygradesc

Logistic Regression

Berbeda dengan **Linear Regression**, **Logistic Regression** digunakan untuk permasalahan Binary Classification. Jadi output yang dihasilkan adalah diskrit ! (1 atau 0), (ya atau tidak).

Misal, diberikan 2 hal:

1. Unseen data ***x*** yang ingin diprediksi labelnya, yaitu ***y***.
2. Model logistic regression dengan parameter $\boldsymbol{\theta_0, \theta_1, \ldots, \theta_n}$ yang sudah ditentukan.

$$P(y=1 \mid x;\theta) = \sigma(\theta_0 + \theta_1 x_1 + \ldots + \theta_n x_n) = \sigma(\theta_0 + \sum_{i=1}^{n} \theta_i x_i)$$

$$P(y=0 \mid x;\theta) = 1 - P(y=1 \mid x;\theta)$$

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

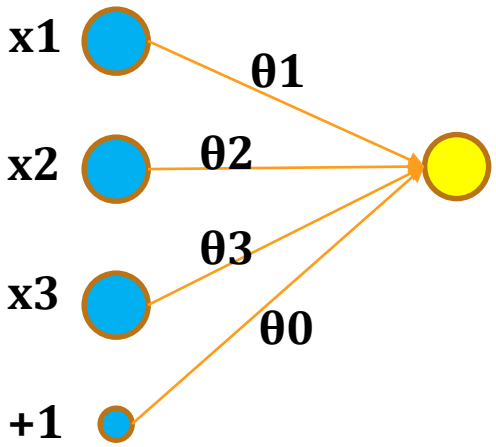
Logistic Regression

Bisa digambarkan dalam bentuk "**single neuron**".

$$P(y=1|x;\theta) = \sigma(\theta_0 + \theta_1 x_1 + \ldots + \theta_n x_n) = \sigma(\theta_0 + \sum_{i=1}^{n} \theta_i x_i)$$

$$P(y=0|x;\theta) = 1 - P(y=1|x;\theta)$$

$$\sigma(z) = \frac{1}{1+e^{-z}}$$



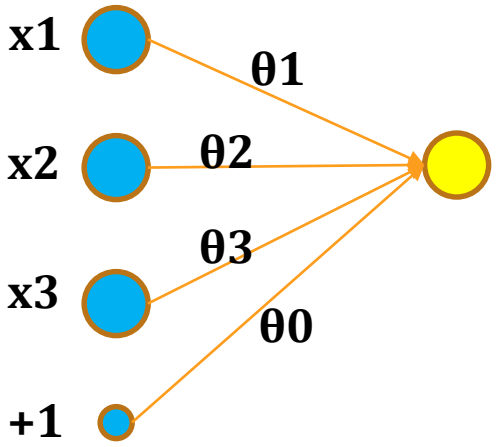
Dengan **fungsi sigmoid** sebagai **activation function**

Logistic Regression

Slide sebelumnya mengasumsikan bahwa parameter **θ** sudah ditentukan.

Bagaimana bila belum ditentukan ?

Kita bisa estimasi parameter **θ** dengan memanfaatkan training data $\{(\mathbf{x^{(1)}}, y^{(1)}), (\mathbf{x^{(2)}}, y^{(2)}), \ldots, (\mathbf{y^{(n)}}, y^{(n)})\}$ yang diberikan (**learning**).



Logistic Regression

Learning

Misal,

$$h_\theta(x) = \sigma(\theta_0 + \theta_1 x_1 + \ldots + \theta_n x_n) = \sigma(\theta_0 + \sum_{i=1}^{n} \theta_i x_i)$$

Probabilitas masing-masing kelas dapat dituliskan **secara singkat** (persamaan di slide-slide sebelumnya) dengan:

$$P(y \mid x;\theta) = (h_\theta(x))^y . (1 - h_\theta(x))^{1-y} \qquad y \in \{0,1\}$$

Diberikan **m** buah training examples, likelihood:

$$L(\theta) = \prod_{i=1}^{m} P(y^{(i)} \mid x^{(i)};\theta)$$

$$= \prod_{i=1}^{m} (h_\theta(x^{(i)}))^{y^{(i)}} . (1 - h_\theta(x^{(i)}))^{1-y^{(i)}}$$

Logistic Regression

Learning

Dengan **MLE**, kita akan mencari konfigurasi parameter yang memberikan nilai likelihood **maksimal (= nilai log-likelihood maksimal**).

$$l(\theta) = \log L(\theta)$$

$$= \sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))$$

Atau, kita cari parameter yang **meminimalkan** fungsi ***negative log-likelihood (cross-entropy loss)***. Inilah **cost function** kita !

$$J(\theta) = -\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))$$

Logistic Regression

Learning

Agar bisa menggunakan Gradient Descent untuk mencari parameter yang meminimalkan cost function, kita perlu menurunkan:

$$\frac{\partial}{\partial \theta_i} J(\theta)$$

Dapat ditunjukkan bahwa untuk **sebuah training example (x,y)**:

$$\frac{\partial}{\partial \theta_j} J(\theta) = (h_\theta(x) - y)\, x_j$$

Jadi, untuk update nilai parameter di tiap iterasi **untuk semua example**:

$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})\, x_j^i$$

Logistic Regression

Learning

Batch Gradient Descent untuk learning model

inisialisasi $\theta_1, \theta_2, \ldots, \theta_n$

while not converged :

$$\theta_1 := \theta_1 - \alpha \sum_{i=1}^{m} (-h_\theta(x^{(i)}) - y^{(i)})\, x_1^i$$

$$\theta_2 := \theta_2 - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})\, x_2^i$$

$\ldots$

$$\theta_n := \theta_n - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})\, x_n^i$$

Logistic Regression

Learning

Stochastic Gradient Descent: kita bisa membuat progress dan update parameter ketika kita melihat **sebuah training example**.

```
inisialisasi θ1, θ2, …, θn
while not converged :
        for i := 1 to m do :
```

$$\theta_1 := \theta_1 - \alpha\,(y^{(i)} - h_\theta(x^{(i)}))\,x_1^i$$
$$\theta_2 := \theta_2 - \alpha\,(y^{(i)} - h_\theta(x^{(i)}))\,x_2^i$$
$$\dots$$
$$\theta_n := \theta_n - \alpha\,(y^{(i)} - h_\theta(x^{(i)}))\,x_n^i$$

Logistic Regression

Learning

Mini-Batch Gradient Descent untuk learning model: Daripada kita gunakan **sebuah sample** untuk update (seperti **online learning**), gradient dihitung dengan cara rata-rata/sum dari sebuah **mini-batch** sample (misal, 32 atau 64 sample).

Batch Gradient Descent: gradient dihitung untuk **semua** sample!

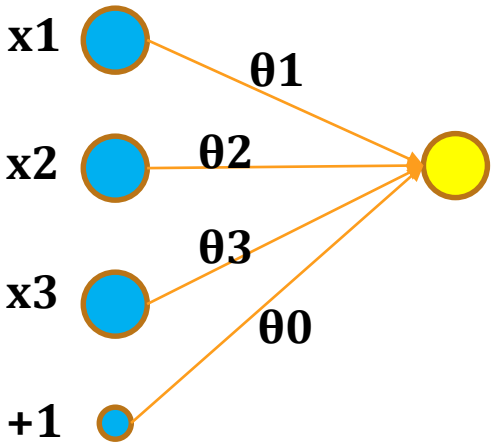
- Untuk sekali step, terlalu besar komputasinya

Online Learning: gradient dihitung untuk **sebuah** sample!

- Terkadang noisy
- Update step sangat kecil

Multilayer Neural Network (Multilayer Perceptron)

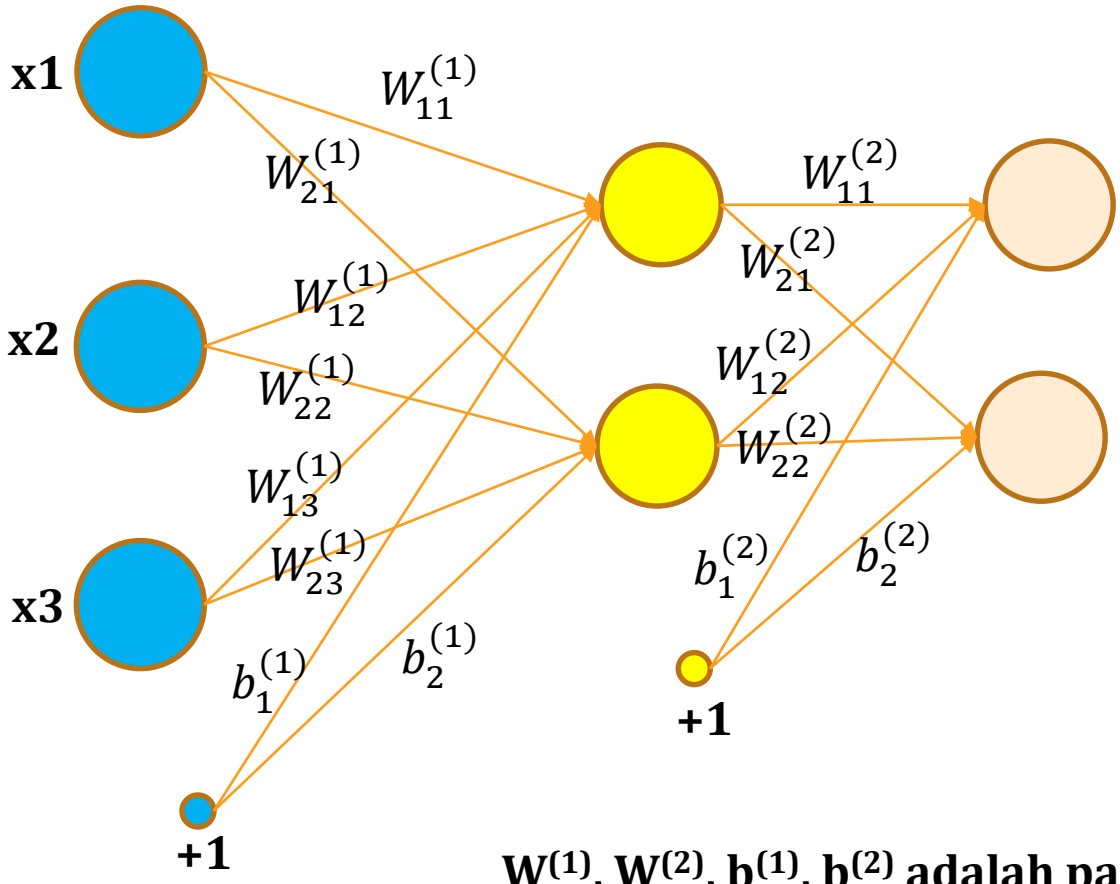
Logistic Regression sebenarnya bisa dianggap sebagai Neural Network dengan *beberapa unit di layer input*, *satu unit di layer output*, dan *tanpa hidden layer*.



Dengan **fungsi sigmoid** sebagai **activation function**

Multilayer Neural Network (Multilayer Perceptron)

Misal, ada 3-layer NN, dengan 3 input unit, 2 hidden unit, dan 2 output unit.



$W^{(1)}$, $W^{(2)}$, $b^{(1)}$, $b^{(2)}$ adalah parameter !

Multilayer Neural Network (Multilayer Perceptron)

Seandainya parameter sudah diketahui, bagaimana caranya memprediksi label dari input (**klasifikasi**) ?

Dari contoh sebelumnya, ada **2 unit di output layer**. Kondisi ini biasanya digunakan untuk binary classification. Unit pertama menghasilkan probabilitas untuk pertama, dan unit kedua menghasilkan probabilitas untuk kelas kedua.

Kita perlu melakukan proses **feed-forward** untuk menghitung nilai yang dihasilkan di output layer.

Multilayer Neural Network (Multilayer Perceptron)

Misal, untuk activation function, kita gunakan fungsi **hyperbolic tangent**.

$$f(x) = \tanh(x)$$

Untuk menghitung **output di hidden layer**:

$$z_1^{(2)} = W_{11}^{(1)} x_1 + W_{12}^{(1)} x_2 + W_{13}^{(1)} x_3 + b_1^{(1)}$$
$$z_2^{(2)} = W_{21}^{(1)} x_1 + W_{22}^{(1)} x_2 + W_{23}^{(1)} x_3 + b_2^{(1)}$$

$$a_1^{(2)} = f(z_1^{(2)})$$
$$a_2^{(2)} = f(z_2^{(2)})$$

Ini hanyalah perkalian matrix !

$$z^{(2)} = W^{(1)} x + b^{(1)} = \begin{bmatrix} W_{11}^{(1)} & W_{12}^{(1)} & W_{13}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} & W_{23}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix}$$

Multilayer Neural Network (Multilayer Perceptron)

Jadi, Proses **feed-forward** secara keseluruhan hingga menghasilkan output di kedua output unit adalah:

$$z^{(2)} = W^{(1)}x + b^{(1)}$$
$$a^{(2)} = f(z^{(2)})$$
$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$
$$h_{W,b}(x) = a^{(3)} = softmax(z^{(3)})$$

Multilayer Neural Network (Multilayer Perceptron)

Learning

Ada Cost function yang berupa **cross-entropy loss**:

m adalah banyaknya training examples.

$$J(W,b) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j} y_{i,j}\,\log(p_{i,j}) + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(W_{ji}^{(l)})^2$$

Regularization terms

Berarti, cost function untuk satu sample **(x,y)** adalah:

$$J(W,b;x,y) = -\sum_{j} y_j \log\left(h_{W,b}(x_j)\right) = -\sum_{j} y_j \log\left(p_j\right)$$

Multilayer Neural Network (Multilayer Perceptron)

Learning

Namun, kali ini, Cost function kita adalah **squared-error**:

m adalah banyaknya training examples.

$$J(W,b) = \frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\left\|h_{W,b}(x^{(i)}) - y^{(i)}\right\|^2\right) + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(W_{ji}^{(l)})^2$$

Regularization terms

Berarti, cost function untuk satu sample **(x,y)** adalah:

$$J(W,b;x,y) = \frac{1}{2}\left\|h_{W,b}(x^{(i)}) - y^{(i)}\right\|^2 = \frac{1}{2}\sum_j\left(h_{W,b}(x_j^{(i)}) - y_j^{(i)}\right)^2$$

Multilayer Neural Network (Multilayer Perceptron)

Learning

Batch Gradient Descent

```
inisialisasi W, b
while not converged :
```

Bagaimana cara menghitung gradient ??

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b)$$

$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W,b)$$

Multilayer Neural Network (Multilayer Perceptron)

Learning

Misal, **(x, y)** adalah **sebuah** training sample, dan **dJ(W,b;x,y)** adalah turunan parsial terkait sebuah sample **(x,y)**.

dJ(W,b;x,y) menentukan *overall partial derivative* **dJ(W,b):**

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b) = \left[\frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x^{(i)},y^{(i)}) \right] + \lambda W_{ij}^{(l)}$$

$$\frac{\partial}{\partial b_{i}^{(l)}} J(W,b) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial b_{i}^{(l)}} J(W,b;x^{(i)},y^{(i)})$$

Dihitung dengan teknik **BackPropagation**

Multilayer Neural Network (Multilayer Perceptron)

Learning

Back-Propagation

1. Jalankan proses **feed-forward**
2. Untuk setiap output unit **i** pada layer $\mathbf{n_l}$ (output layer)

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} J(W, b; x, y) = (a_i^{(n_l)} - y_i) \cdot f'(z_i^{(n_l)})$$

3. ***l*** = $\mathbf{n_l}$ - 1, $\mathbf{n_l}$ - 2, … 2

   Untuk setiap node **i** di layer ***l***

$$\delta_i^{(l)} = (\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)}) \cdot f'(z_i^{(l)})$$

4. Finally..

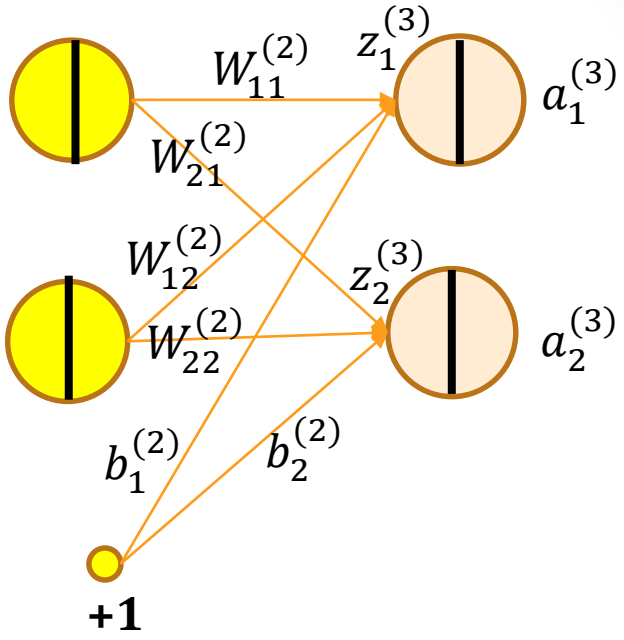
$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y) = a_j^{(l)} \delta_i^{(l+1)} \qquad \frac{\partial}{\partial b_i^{(l)}} J(W, b; x, y) = \delta_i^{(l+1)}$$

Multilayer Neural Network (Multilayer Perceptron)

Learning

Back-Propagation

Contoh hitung gradient di output ...



$$J(W,b;x,y) = \frac{1}{2}\left(a_1^{(3)} - y_1\right)^2 + \frac{1}{2}\left(a_2^{(3)} - y_2\right)^2$$

$$\frac{\partial J}{\partial a_1^{(3)}} = \left(a_1^{(3)} - y_1\right)$$

$$\frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} = \frac{\partial\left(f\left(z_1^{(3)}\right)\right)}{\partial z_1^{(3)}} = f'\left(z_1^{(3)}\right)$$

$$z_1^{(3)} = W_{11}^{(2)} \cdot a_1^{(2)} + W_{12}^{(2)} \cdot a_2^{(2)} + b_1^{(2)}$$

$$\frac{\partial z_1^{(3)}}{\partial W_{12}^{(2)}} = a_2^{(2)}$$

$$\frac{\partial}{\partial W_{12}^{(2)}} J(W,b;x,y) = \frac{\partial J}{\partial a_1^{(3)}} \times \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \times \frac{\partial z_1^{(3)}}{\partial W_{12}^{(2)}}$$

$$= \left(a_1^{(3)} - y_1\right) \cdot f'(z_1^{(3)}) \cdot a_2^{(2)}$$

Sensitivity – Jacobian Matrix

The **Jacobian J** is the matrix of partial derivatives of the network output vector **y** with respect to the input vector **x.**

$$J = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix} \qquad J_{k,i} = \frac{\partial y_k}{\partial x_i}$$

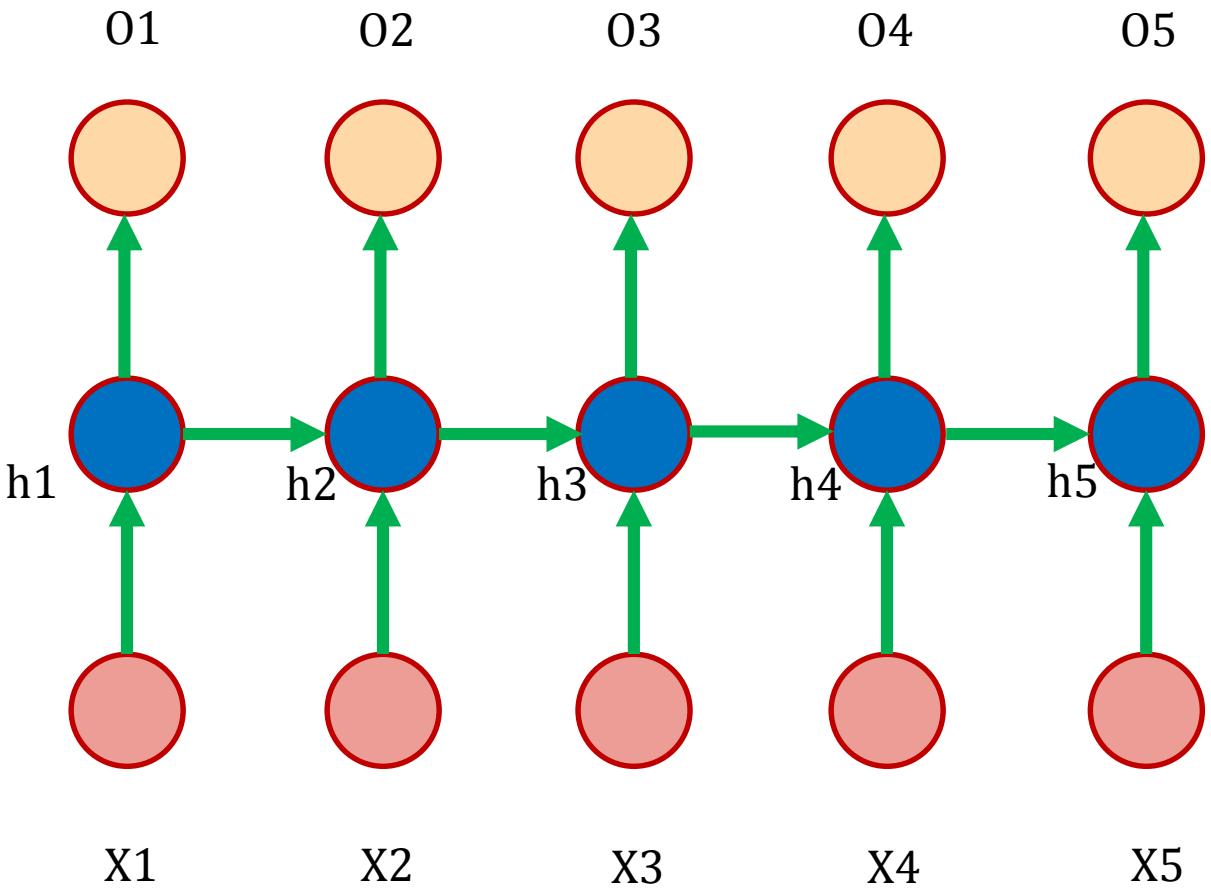
These derivatives measure **the relative sensitivity of the outputs to small changes in the inputs**, and can therefore be used, for example, to detect irrelevant inputs.

Alex Graves, Supervised Sequence Labelling with Recurrent Neural Networks

More Complex Neural Networks
(Neural Network Architectures)

Recurrent Neural Networks (Vanilla RNN, LSTM, GRU)
Attention Mechanisms
Recursive Neural Networks

Recurrent Neural Networks



One of the famous Deep Learning Architectures in the NLP community

Recurrent Neural Networks (RNNs)

Kita biasanya menggunakan **RNNs** untuk:

- Memproses **Sequences**
- Menghasilkan **Sequences**
- …
- Intinya … ada **Sequences**

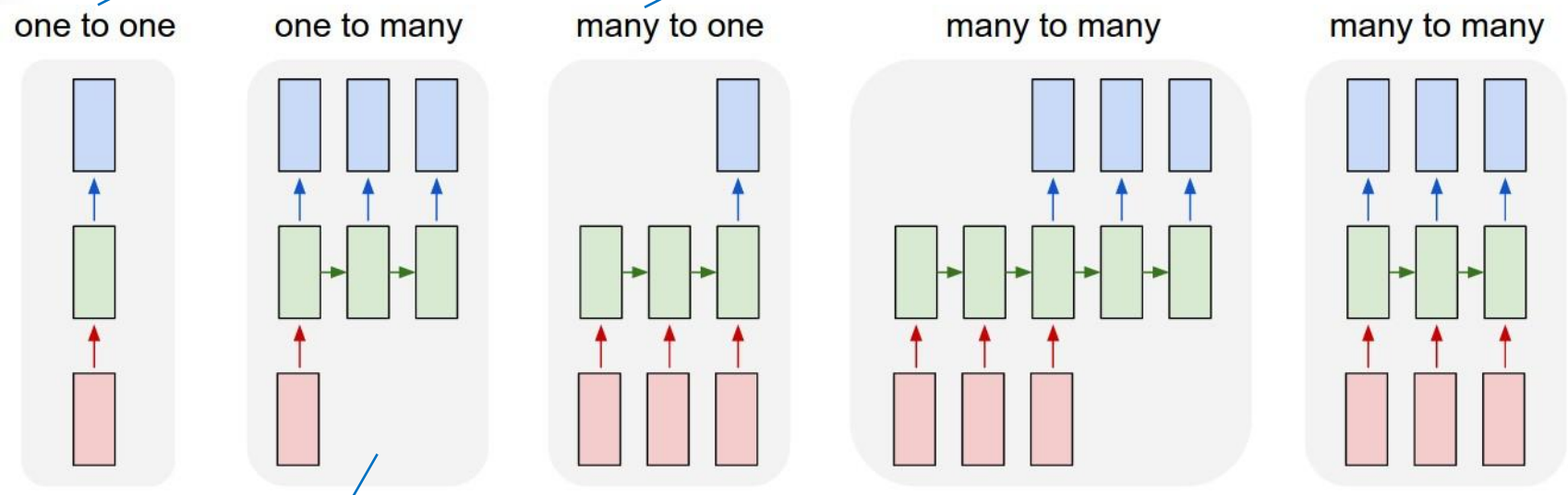
Sequences biasanya:

- Urutan **kata-kata**
- Urutan **kalimat-kalimat**
- **Signal**
- **Suara**
- **Video** (Sequence of Images)
- …

Recurrent Neural Networks (RNNs)

Not RNNs
(Vanilla Feed-Forward NNs)

Sequence Input
(e.g. Sentence Classification)

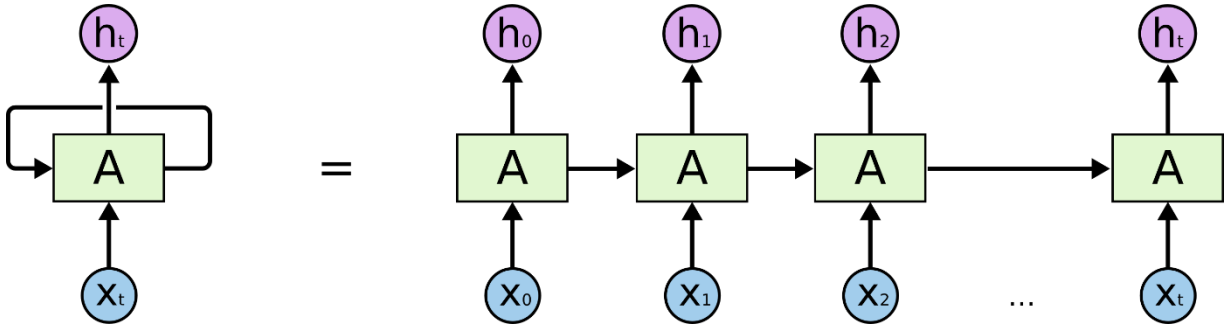


Sequence Output
(e.g. Image Captioning)

Sequence Input/Output
(e.g. Machine Translation)

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Recurrent Neural Networks (RNNs)



RNNs combine the input vector with their state vector with a fixed (but learned) function to produce a new state vector.

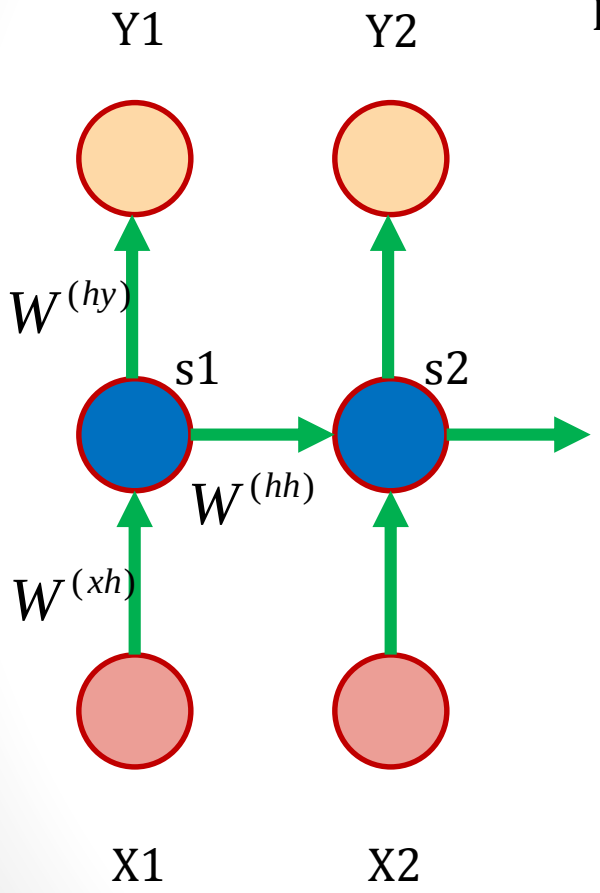
This can, **in programming terms**, be interpreted as running a fixed program with certain inputs and some internal variables.

In fact, it is known that RNNs are Turing-Complete in the sense that they can simulate arbitrary programs (**with proper weights**).

http://binds.cs.umass.edu/papers/1995_Siegelmann_Science.pdf
http://karpathy.github.io/2015/05/21/rnn-effectiveness/
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent Neural Networks (RNNs)

Misal, ada **I** input unit, **K** output unit, dan **H** hidden unit (state).



Komputasi RNNs untuk **satu sample**:

$$h_t \in R^{H\times 1} \quad x_t \in R^{I\times 1} \quad y_t \in R^{K\times 1}$$

$$W^{(xh)} \in R^{H\times I} \quad W^{(hh)} \in R^{H\times H}$$

$$W^{(hy)} \in R^{K\times H}$$

$$h_t = W^{(xh)} \cdot x_t + W^{(hh)} \cdot s_{t-1}$$

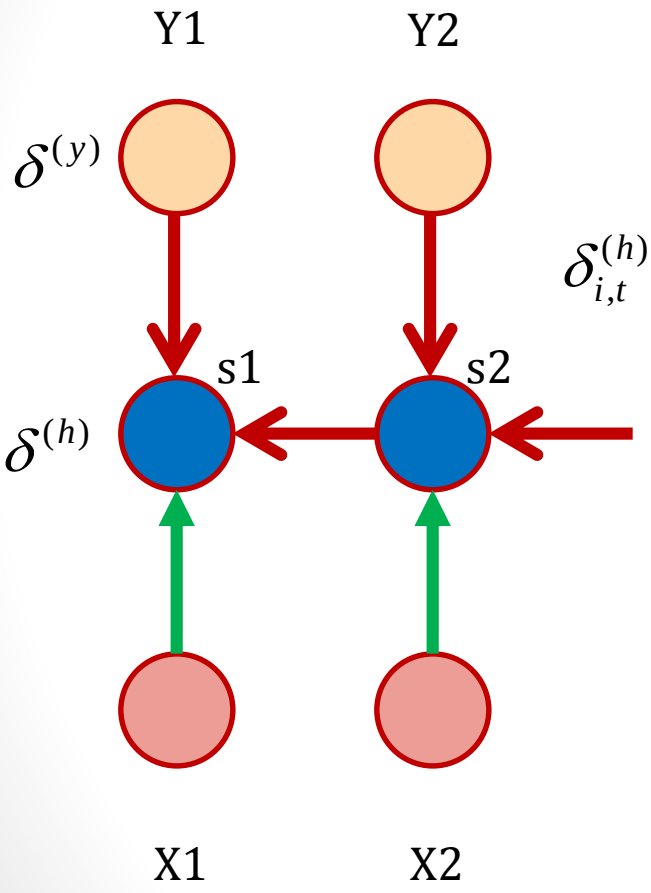
$$s_t = \tanh(h_t)$$

$$y_t = W^{(hy)} \cdot s_t$$

$$h_0 = 0$$

Recurrent Neural Networks (RNNs)

Back Propagation Through Time (BPTT)



The loss function depends on the activation of the hidden layer not only through its influence on the output layer.

$$\delta_{i,t}^{(h)} = \left(\sum_{j=1}^{K} W_{i,j}^{(hy)} \cdot \delta_{j,t}^{(y)} + \sum_{n=1}^{H} W_{i,n}^{(hh)} \cdot \delta_{n,t+1}^{(h)} \right) \cdot f'\left(h_{i,t}\right)$$

Di output: $\delta_{i,t}^{(y)} = \dfrac{\partial L_t}{\partial y_{i,t}}$

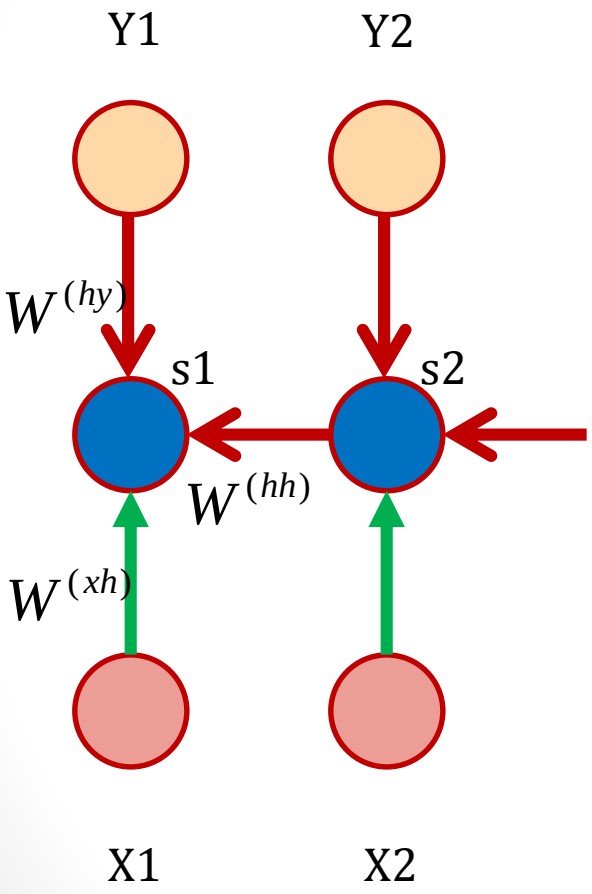
Di setiap step, kecuali paling kanan:

$$\delta_{i,t}^{(h)} = \frac{\partial L_t}{\partial h_{i,t}} \qquad \delta_{i,T+1}^{(h)} = 0$$

Alex Graves, Supervised Sequence Labelling with Recurrent Neural Networks

Recurrent Neural Networks (RNNs)

Back Propagation Through Time (BPTT)



The same weights are reused at every timestep, we sum over the whole sequence to get the derivatives with respect to the network weights.

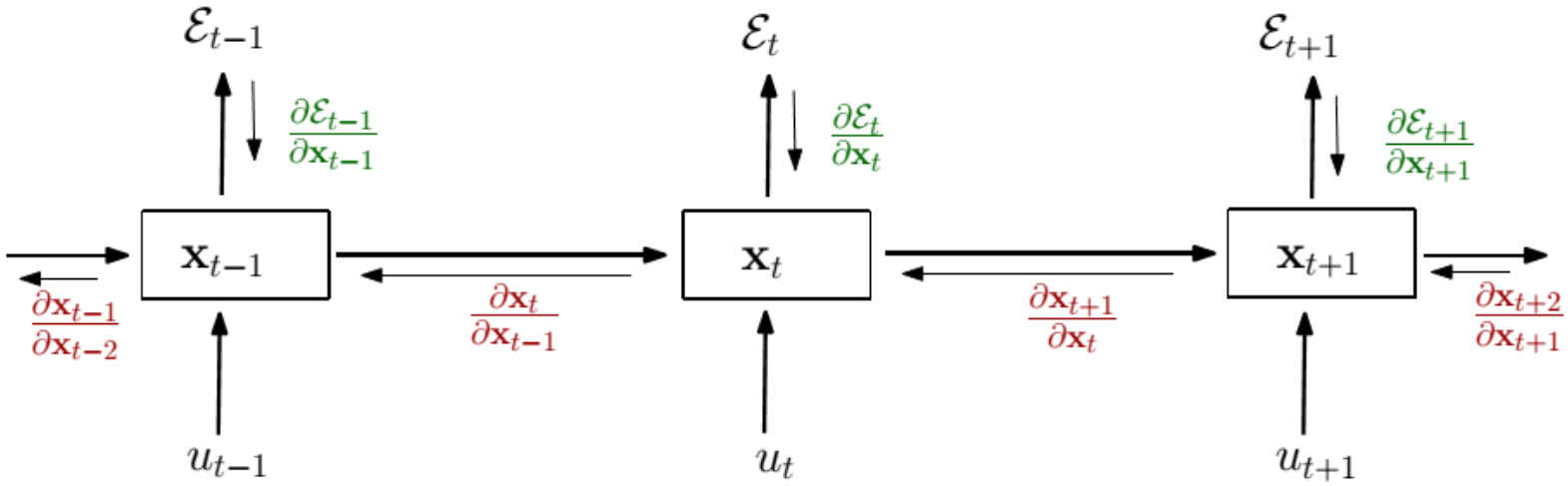
$$\frac{\partial L}{\partial W_{i,j}^{(hh)}} = \sum_{t=1}^{T} \delta_{j,t}^{(h)} \cdot s_{i,t-1}$$

$$\frac{\partial L}{\partial W_{i,j}^{(hy)}} = \sum_{t=1}^{T} \delta_{j,t}^{(y)} \cdot s_{i,t}$$

$$\frac{\partial L}{\partial W_{i,j}^{(xh)}} = \sum_{t=1}^{T} \delta_{j,t}^{(h)} \cdot x_{i,t}$$

Recurrent Neural Networks (RNNs)

Back Propagation Through Time (BPTT)



$$\frac{\partial \mathcal{E}}{\partial \theta} = \sum_{1 \le t \le T} \frac{\partial \mathcal{E}_t}{\partial \theta}$$

$$\frac{\partial \mathcal{E}_t}{\partial \theta} = \sum_{1 \le k \le t} \left(\frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t} \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} \frac{\partial^+ \mathbf{x}_k}{\partial \theta} \right)$$

$$\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} = \prod_{t \ge i > k} \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_{i-1}} = \prod_{t \ge i > k} \mathbf{W}_{rec}^T diag(\sigma'(\mathbf{x}_{i-1}))$$

Pascanu et al., On the difficulty of training Recurrent Neural Networks, 2013

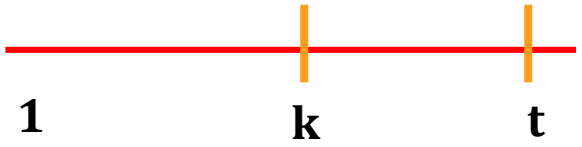
Recurrent Neural Networks (RNNs)

Back Propagation Through Time (BPTT)

Misal, untuk parameter antar state:

Term-term ini disebut **temporal contribution**: bagaimana $\mathbf{W^{(hh)}}$ pada step **k** mempengaruhi cost pada step-step setelahnya **(t > k)**

$$\frac{\partial L_t}{\partial W^{(hh)}} = \sum_{k=1}^{t} \frac{\partial L_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial h_k} \cdot \frac{\partial^+ h_k}{\partial W^{(hh)}}$$



Diputus sampai **k step** ke belakang. Di sini artinya "immediate derivative", yaitu $\mathbf{h_{k-1}}$ dianggap konstan terhadap $\mathbf{W^{(hh)}}$.

$$\frac{\partial h_t}{\partial h_k} = \frac{\partial h_t}{\partial h_{t-1}} \cdot \frac{\partial h_{t-1}}{\partial h_{t-2}} \cdots \frac{\partial h_{k+2}}{\partial h_{k+1}} \cdot \frac{\partial h_{k+1}}{\partial h_k}$$

$$\frac{\partial^+ h_k}{\partial W^{(hh)}} = \frac{\partial^+ \left(W^{(xh)} \cdot x_t + W^{(hh)} \cdot s_{t-1} \right)}{\partial W^{(hh)}} = s_{t-1}$$

Recurrent Neural Networks (RNNs)

Vanishing & Exploding Gradient Problems

Bengio et al., (1994) said that *"the **exploding gradients problem** refers to the large increase in the norm of the gradient during training. Such events are caused by the explosion of the long term components, which can grow exponentially more then short term ones."*

*And "The **vanishing gradients problem** refers to the opposite behaviour, when long term components go exponentially fast to norm 0, making it impossible for the model to learn correlation between temporally distant events."*

Kok bisa terjadi? Coba lihat salah satu temporal component dari sebelumnya:

$$\frac{\partial h_t}{\partial h_k} = \frac{\partial h_t}{\partial h_{t-1}} \cdot \frac{\partial h_{t-1}}{\partial h_{t-2}} \cdots \frac{\partial h_{k+2}}{\partial h_{k+1}} \cdot \frac{\partial h_{k+1}}{\partial h_k}$$

*In the same way a product of **t - k** real numbers can shrink to zero or explode to infinity, so does this product of Matrices. (**Pascanu et al.,**)*

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks

Recurrent Neural Networks (RNNs)

Vanishing & Exploding Gradient Problems

Sequential Jacobian biasa digunakan untuk analisis penggunaan konteks pada RNNs.

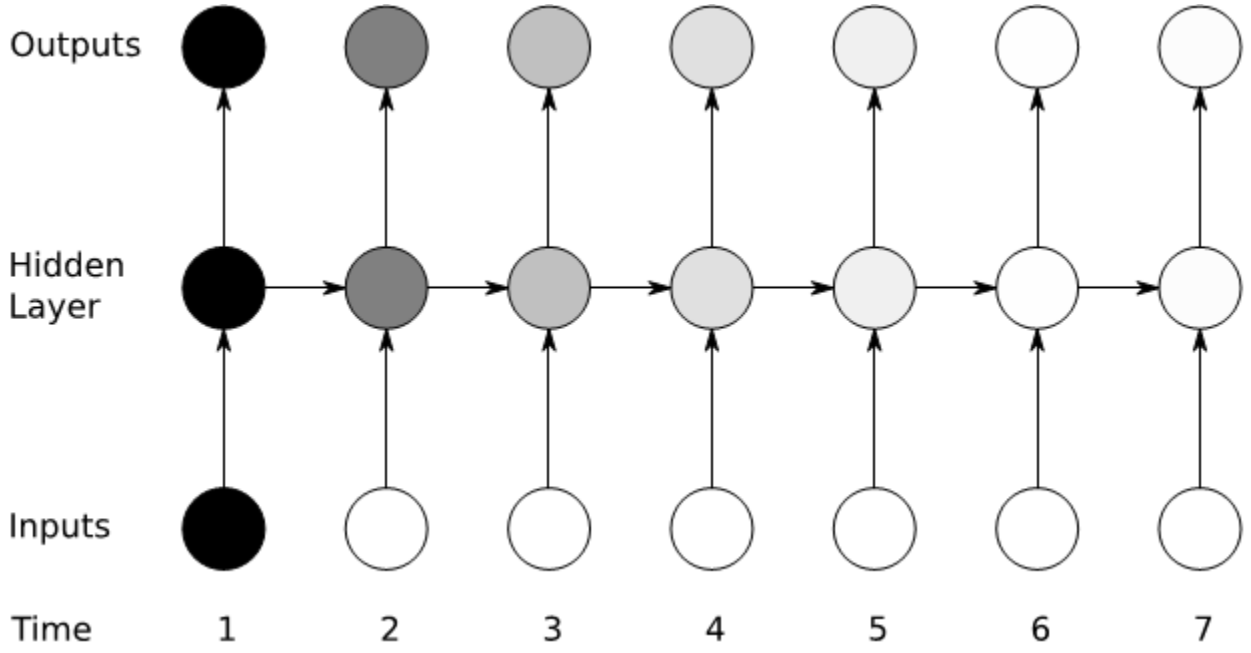


Figure 4.1: **The vanishing gradient problem for RNNs.** The shading of the nodes in the unfolded network indicates their sensitivity to the inputs at time one (the darker the shade, the greater the sensitivity). The sensitivity decays over time as new inputs overwrite the activations of the hidden layer, and the network 'forgets' the first inputs.

Alex Graves, Supervised Sequence Labelling with Recurrent Neural Networks

Recurrent Neural Networks (RNNs)

Solusi untuk Vanishing Gradient Problem

1) Penggunaan ***non-gradient*** *based training algorithms* (Simulated Annealing, Discrete Error Propagation, etc.) **(Bengio et al., 1994)**

2) Definisikan arsitektur baru di dalam **RNN Cell**!, seperti **Long-Short Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997)**.

3) Untuk metode yang lain, silakan merujuk **(Pascanu et al., 2013)**.

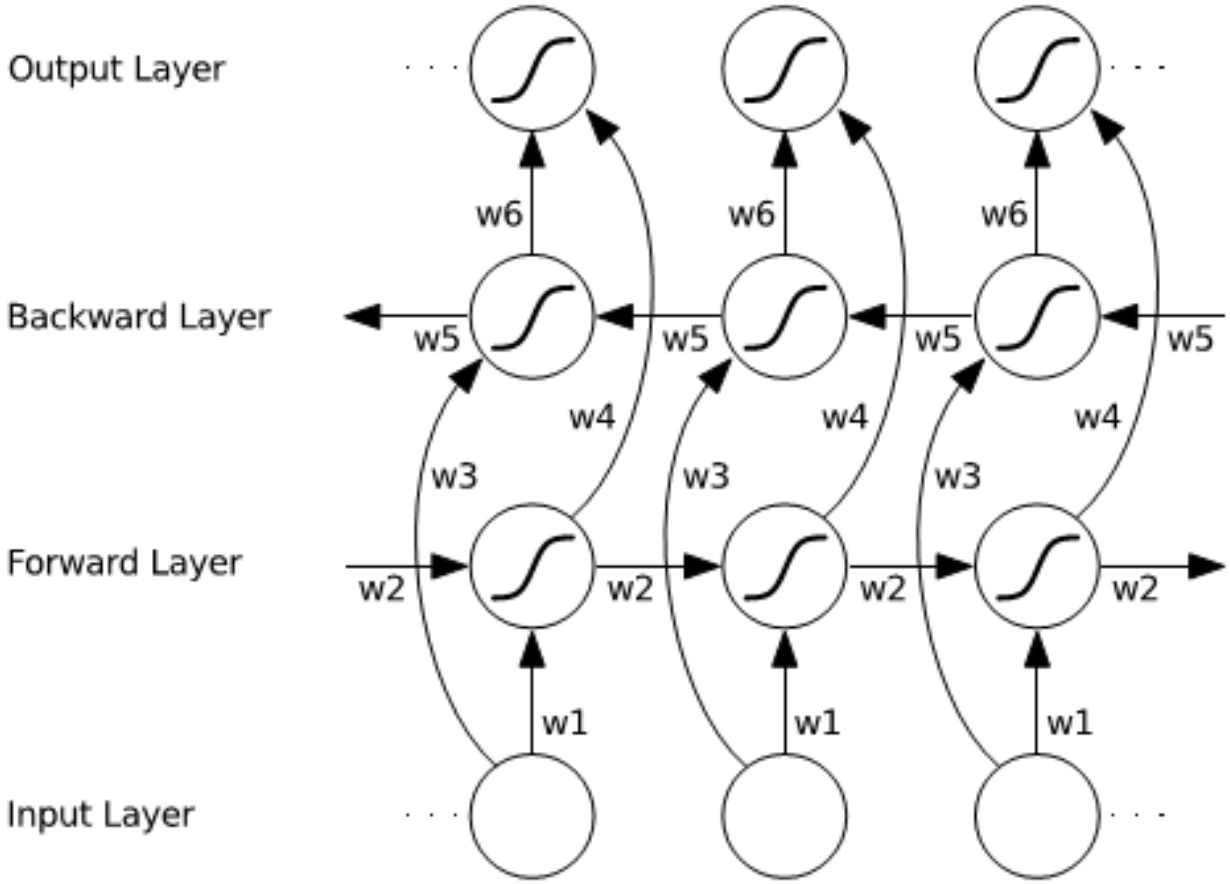
Pascanu et al., On the difficulty of training Recurrent Neural Networks, 2013

S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. Neural Computation,9(8):1735 1780, 1997

Y. Bengio, P. Simard, and P. Frasconi. Learning Long-Term Dependencies with Gradient Descent is Difficult. IEEE Transactions on Neural Networks, 1994

Recurrent Neural Networks (RNNs)

Variant: **Bi-Directional RNNs**



Alex Graves, Supervised Sequence Labelling with Recurrent Neural Networks

Recurrent Neural Networks (RNNs)

Sequential Jacobian (Sensitivity) untuk **Bi-Directional RNNs**

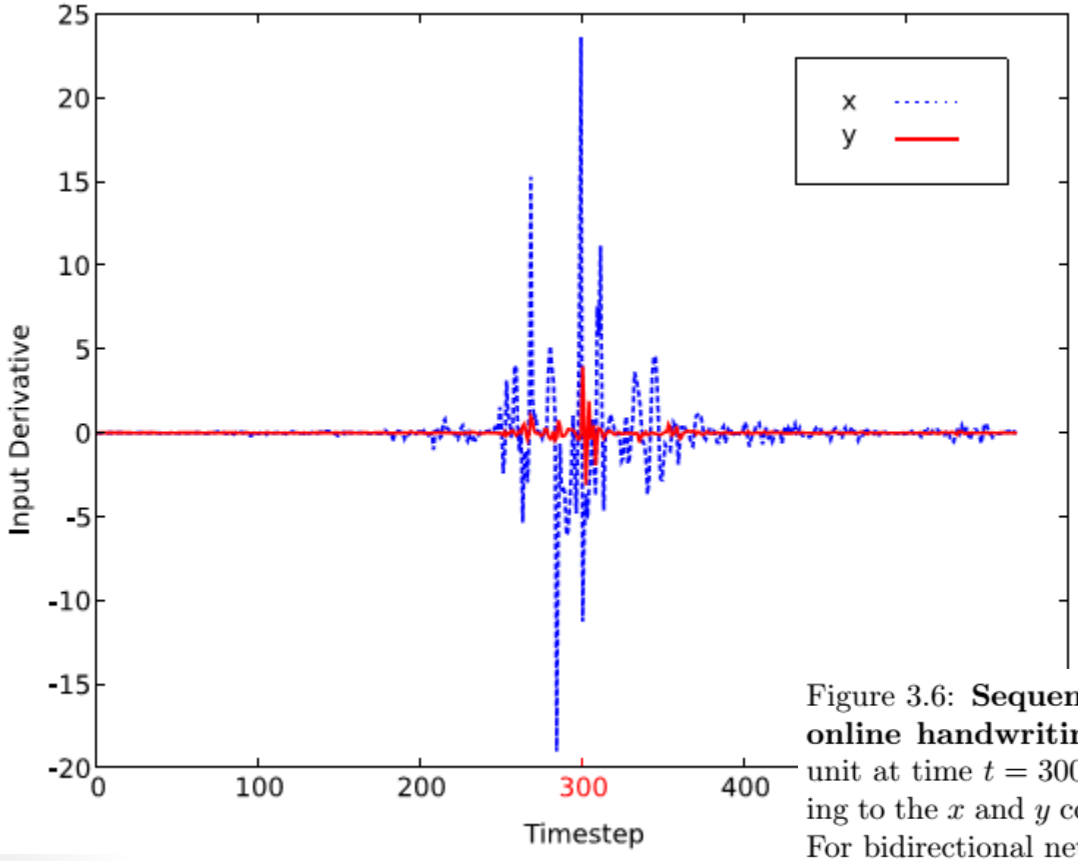


Figure 3.6: **Sequential Jacobian for a bidirectional network during an online handwriting recognition task.** The derivatives of a single output unit at time $t = 300$ are evaluated with respect to the two inputs (corresponding to the x and y coordinates of the pen) at all times throughout the sequence. For bidirectional networks, the magnitude of the derivatives typically forms an 'envelope' centred on t. In this case the derivatives remains large for about 100 timesteps before and after t. The magnitudes are greater for the input corresponding to the x coordinate (blue line) because this has a smaller normalised variance than the y input (x tends to increase steadily as the pen moves from left to right, whereas y fluctuates about a fixed baseline); this does *not* imply that the network makes more use of the x coordinates than the y coordinates.

Alex Graves, Supervised Sequence Labelling with Recurrent Neural Networks

Long-Short Term Memory (LSTM)

1. The LSTM architecture consists of a set of **recurrently connected subnets**, known as **memory blocks**.
2. These blocks can be thought of as a differentiable version of the memory chips in a digital computer.
3. Each block contains:
    1. Self-connected memory cells
    2. Three **multiplicative** units (gates)
        1. Input gates (analogue of write operation)
        2. Output gates (analogue of read operation)
        3. Forget gates ((analogue of reset operation))

S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. Neural Computation,9(8):1735 1780, 1997

Long-Short Term Memory (LSTM)

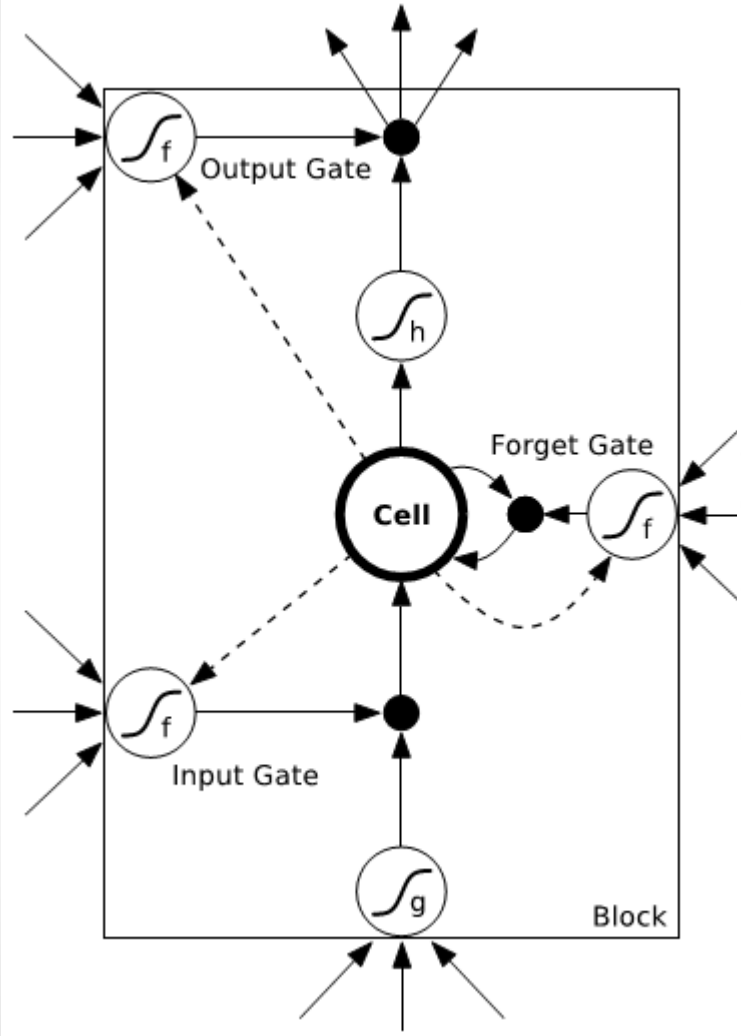
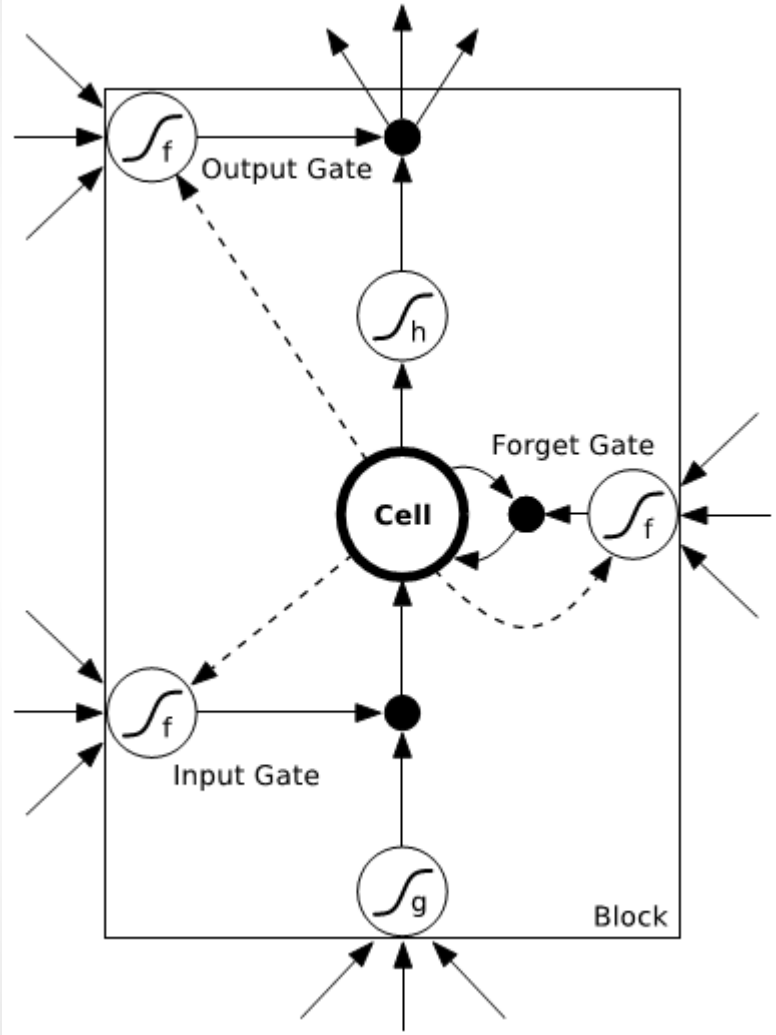


Figure 4.2: **LSTM memory block with one cell.** The three gates are nonlinear summation units that collect activations from inside and outside the block, and control the activation of the cell via multiplications (small black circles). The input and output gates multiply the input and output of the cell while the forget gate multiplies the cell's previous state. No activation function is applied within the cell. The gate activation function 'f' is usually the logistic sigmoid, so that the gate activations are between 0 (gate closed) and 1 (gate open). The cell input and output activation functions ('g' and 'h') are usually tanh or logistic sigmoid, though in some cases 'h' is the identity function. The weighted 'peephole' connections from the cell to the gates are shown with dashed lines. All other connections within the block are unweighted (or equivalently, have a fixed weight of 1.0). The only outputs from the block to the rest of the network emanate from the output gate multiplication.

Alex Graves, Supervised Sequence Labelling with Recurrent Neural Networks

S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. Neural Computation,9(8):1735 1780, 1997

Long-Short Term Memory (LSTM)



The **multiplicative gates** allow LSTM memory cells to store and access information over long periods of time, thereby **mitigating the vanishing gradient problem**.

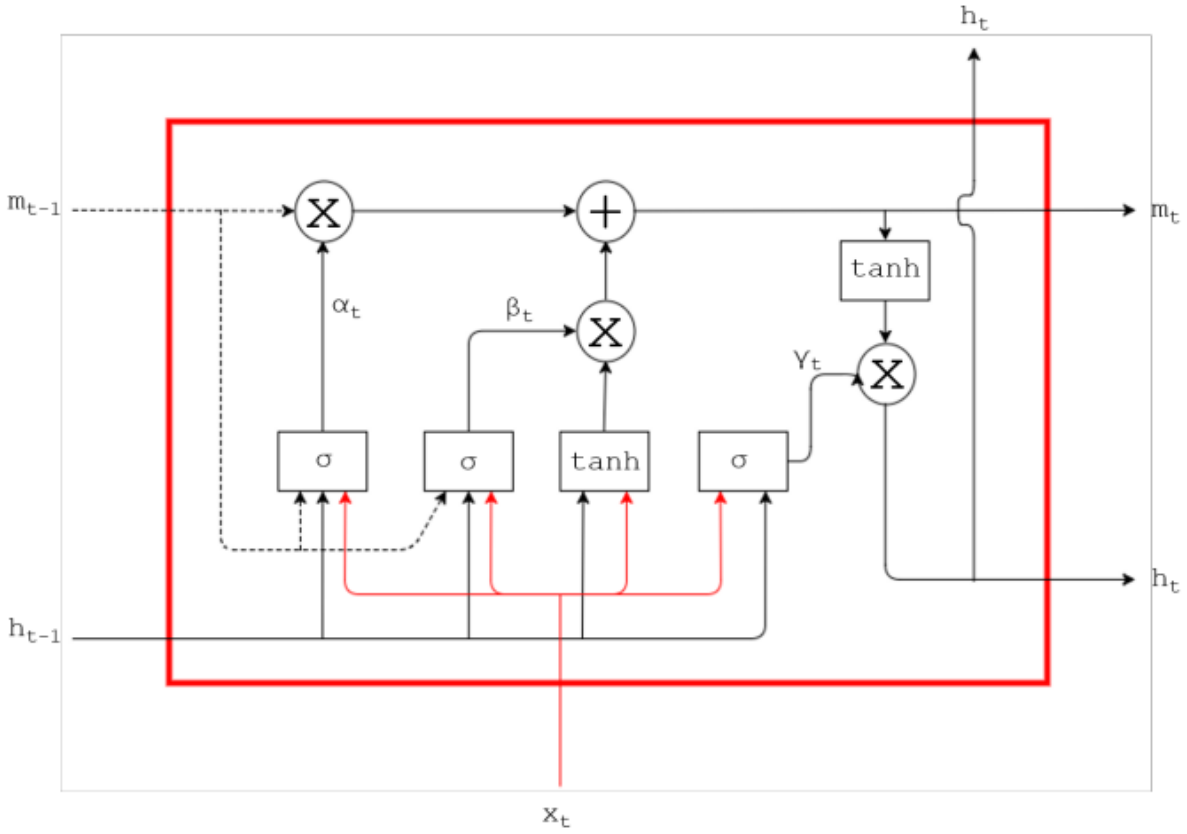
For example, as long as the input gate remains closed (i.e. has an activation near 0), the activation of the cell will not be overwritten by the new inputs arriving in the network, and can therefore be made available to the net much later in the sequence, by opening the output gate.

Alex Graves, Supervised Sequence Labelling with Recurrent Neural Networks

S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. Neural Computation,9(8):1735 1780, 1997

Long-Short Term Memory (LSTM)

Visualisasi lain dari satu cell di LSTM



S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. Neural Computation,9(8):1735 1780, 1997

Long-Short Term Memory (LSTM)

Komputasi di LSTM

$$m_t = \alpha_t(\times)m_{t-1} + \beta_t(\times)f(x_t, t-1)$$

$$h_t = \gamma_t(\times)tanh(m_t)$$

$$f(x_t, t-1) = tanh(W_{xm} \cdot x_t + W_{hm} \cdot h_{t-1})$$

α_t, β_t dan γ_t merupakan *gates*:

(a) *Forget gates*: $\alpha_t = \sigma(W_{x\alpha} \cdot x_t + W_{h\alpha} \cdot h_{t-1} + W_{m\alpha} \cdot m_{t-1})$

(b) *Input gates*: $\beta_t = \sigma(W_{x\beta} \cdot x_t + W_{h\beta} \cdot h_{t-1} + W_{m\beta} \cdot m_{t-1})$

(c) *Output gates*: $\gamma_t = \sigma(W_{x\gamma} \cdot x_t + W_{h\gamma} \cdot h_{t-1} + W_{m\gamma} \cdot m_{t-1})$

S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. Neural Computation,9(8):1735 1780, 1997

Long-Short Term Memory (LSTM)

Preservation of Gradient Information pada LSTM

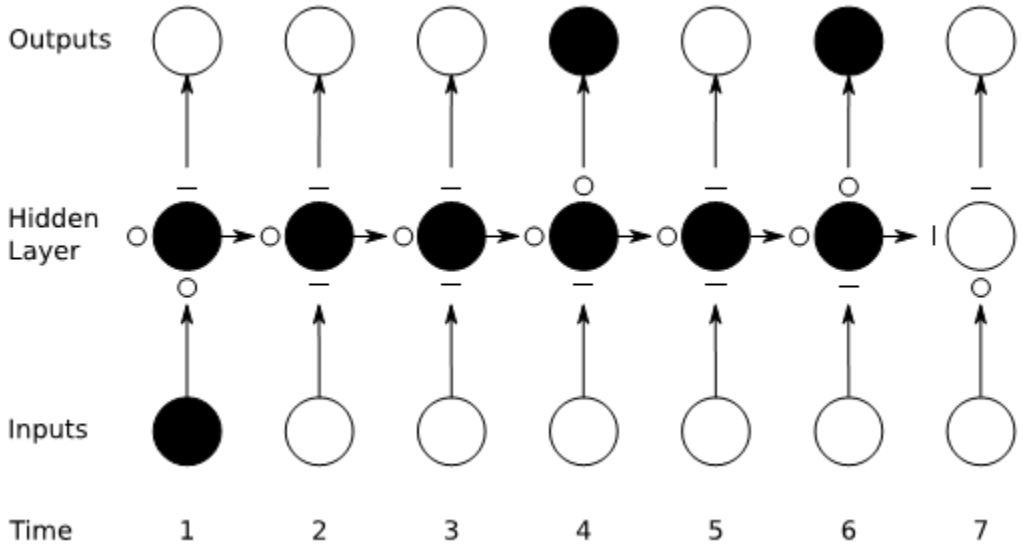


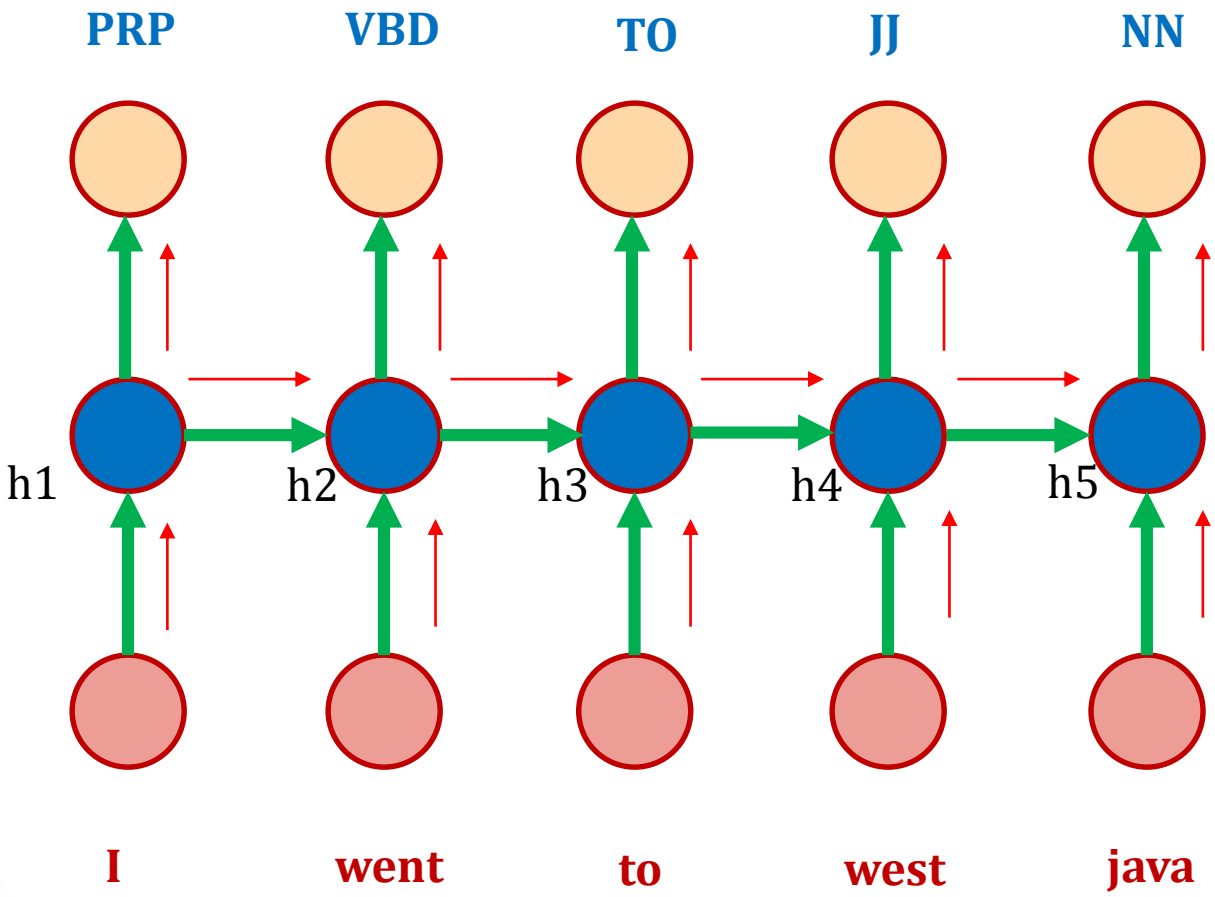
Figure 4.4: **Preservation of gradient information by LSTM.** As in Figure 4.1 the shading of the nodes indicates their sensitivity to the inputs at time one; in this case the black nodes are maximally sensitive and the white nodes are entirely insensitive. The state of the input, forget, and output gates are displayed below, to the left and above the hidden layer respectively. For simplicity, all gates are either entirely open ('O') or closed ('—'). The memory cell 'remembers' the first input as long as the forget gate is open and the input gate is closed. The sensitivity of the output layer can be switched on and off by the output gate without affecting the cell.

Alex Graves, Supervised Sequence Labelling with Recurrent Neural Networks

S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. Neural Computation,9(8):1735 1780, 1997

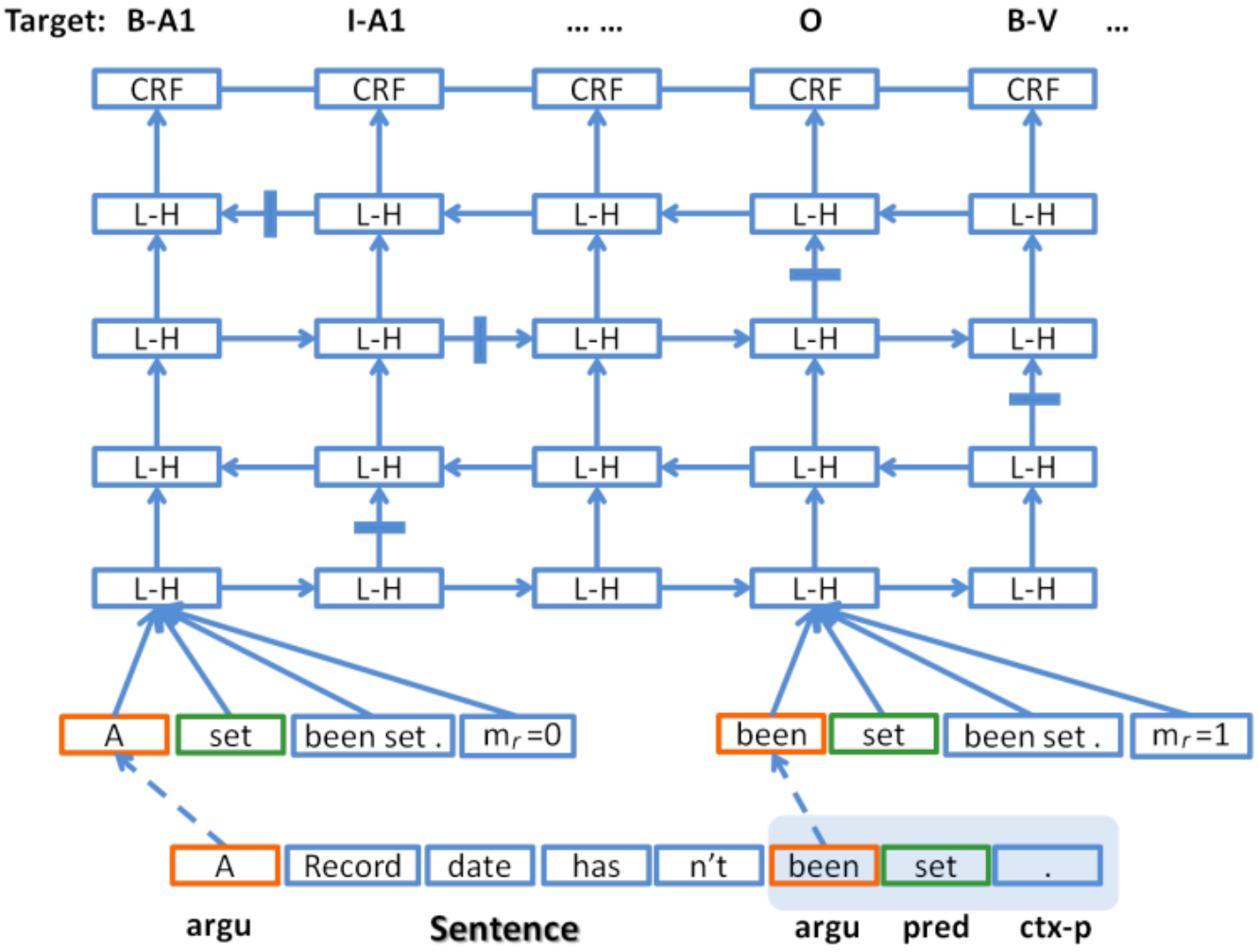
Example: RNNs for POS Tagger

(Zennaki, 2015)



LSTM + CRF for Semantic Role Labeling

(Zhou and Xu, ACL 2015)



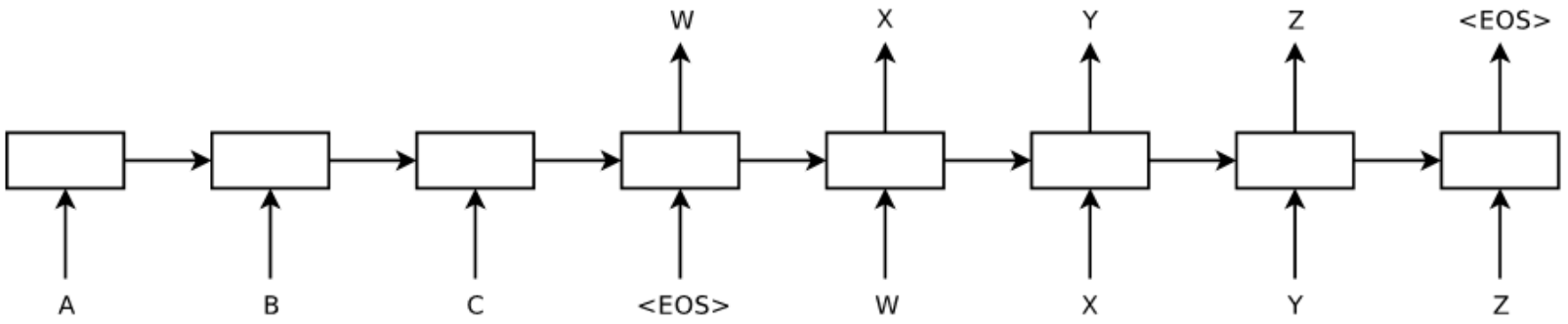
Attention Mechanism

A potential issue with this encoder–decoder approach is that a neural network needs to be able to compress all the necessary information of a source sentence into a fixed-length vector. This may make it difficult for the neural network to cope with long sentences, especially those that are longer than the sentences in the training corpus.

Dzmitry Bahdanau, et al., Neural machine translation by jointly learning to align and translate, 2015

Attention Mechanism

Neural Translation Model **TANPA** Attention Mechanism

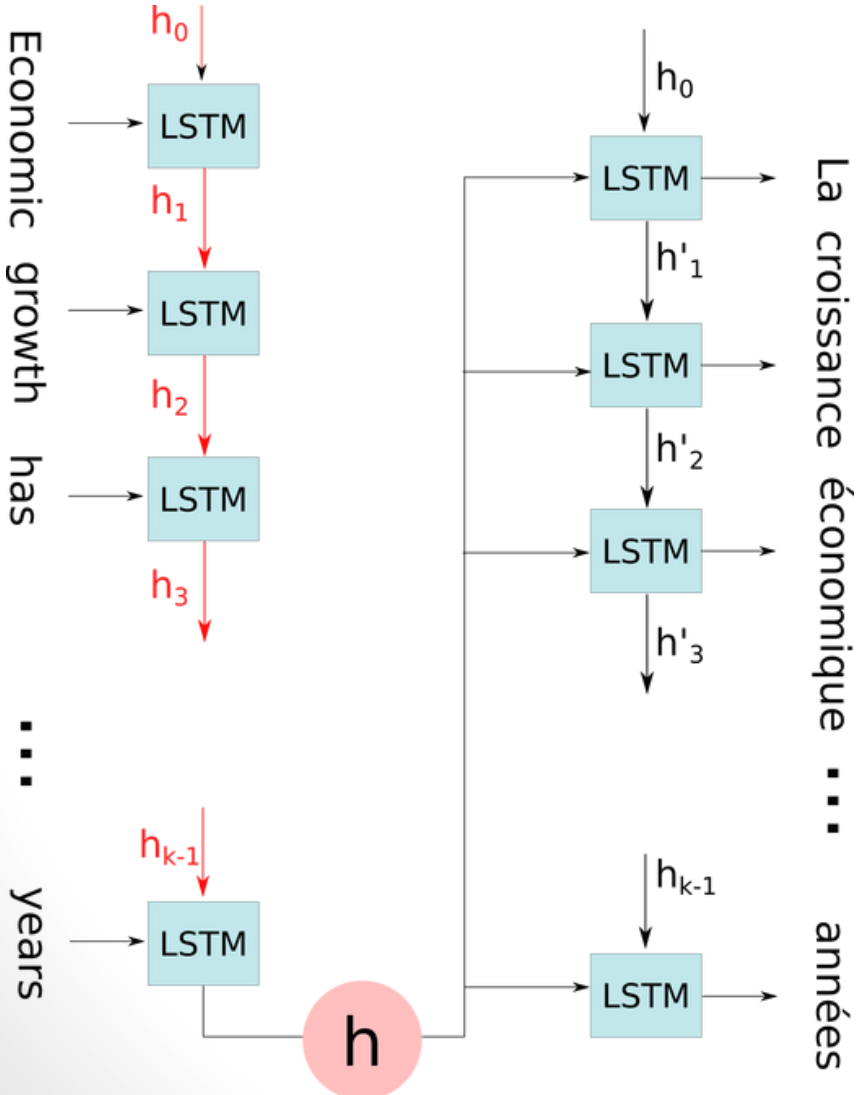


Sutkever, Ilya et al., Sequence to Sequence Learning with Neural Networks, NIPS 2014.

https://blog.heuritech.com/2016/01/20/attention-mechanism/

Attention Mechanism

Neural Translation Model **TANPA** Attention Mechanism



Sutkever, Ilya et al., Sequence to Sequence Learning with Neural Networks, NIPS 2014.

Attention Mechanism

Mengapa Perlu **Attention Mechanism**?

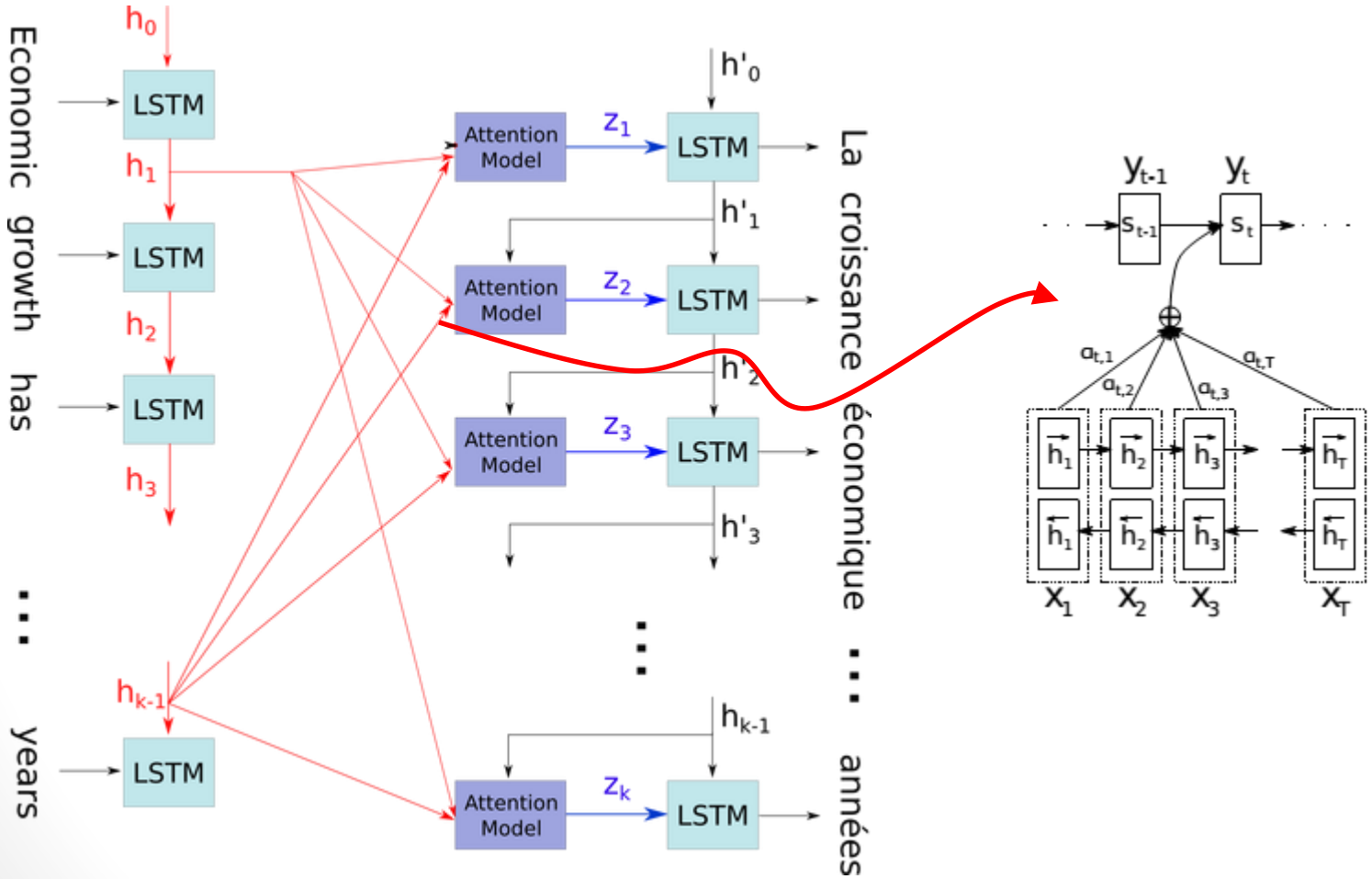
- *Each time the proposed model generates a word in a translation, it (soft-)searches for a set of positions in a source sentence where the most relevant information is concentrated. The model then predicts a target word based on the context vectors associated with these source positions and all the previous generated target words.*
- *… it encodes the input sentence into a sequence of vectors and chooses a subset of these vectors adaptively while decoding the translation. This frees a neural translation model from having to squash all the information of a source sentence, regardless of its length, into a fixed-length vector.*

Dzmitry Bahdanau, et al., Neural machine translation by jointly learning to align and translate, 2015

https://blog.heuritech.com/2016/01/20/attention-mechanism/

Attention Mechanism

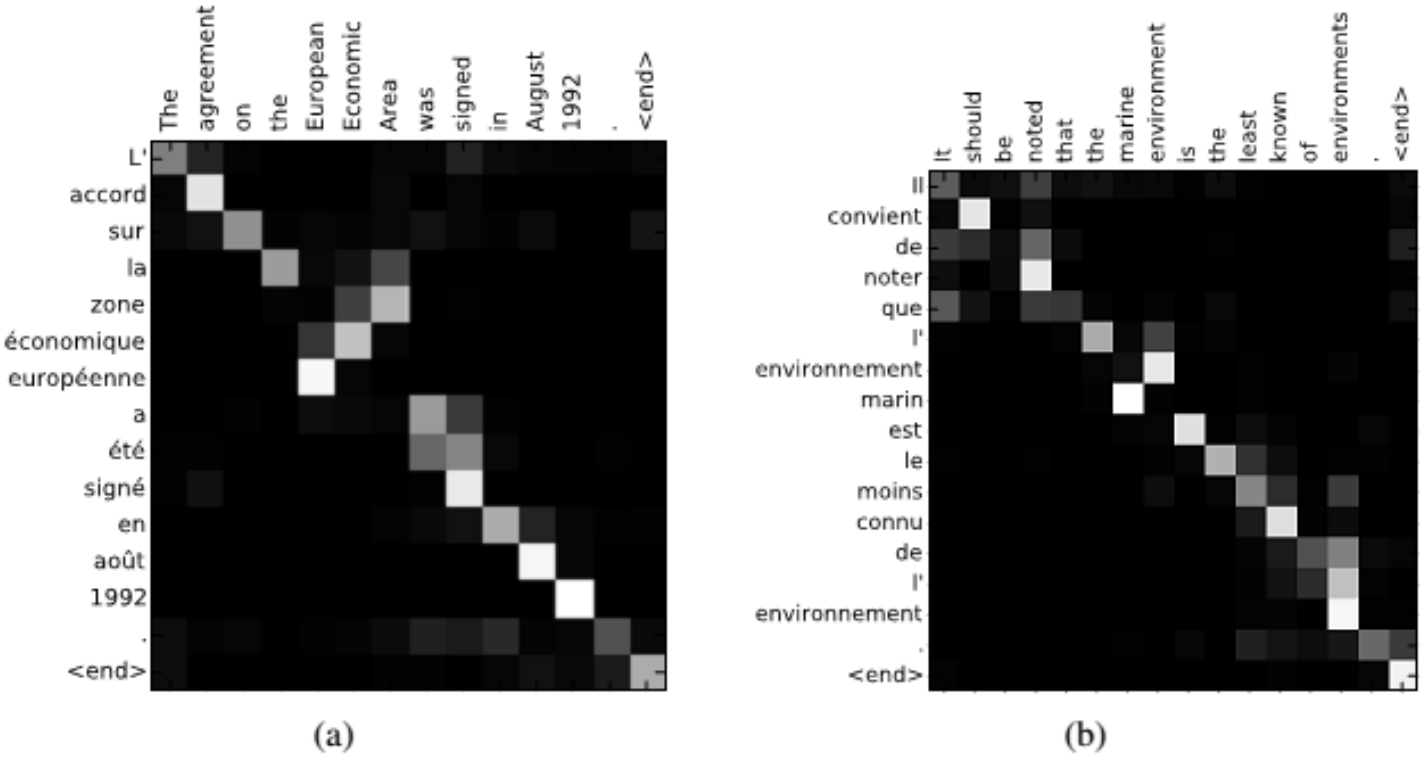
Neural Translation Model – dengan Attention Mechanism



Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio, **Neural Machine Translation by Jointly Learning to Align and Translate, arXiv:1409.0473,** *2016*

Attention Mechanism

Neural Translation Model – dengan Attention Mechanism

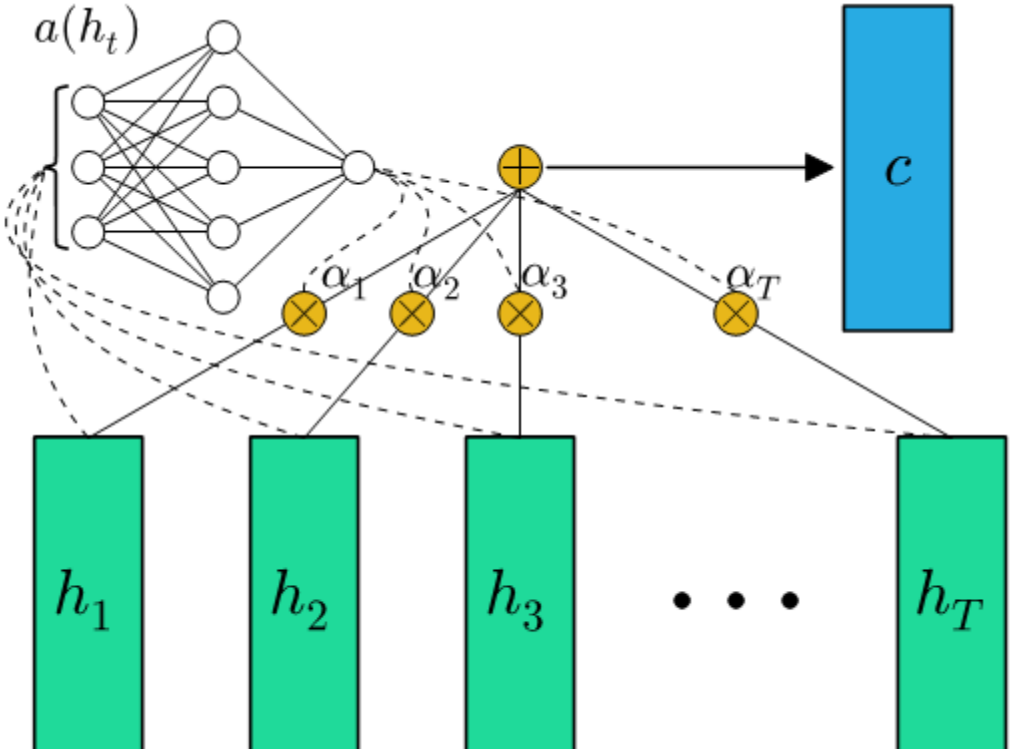


Cell merepresentasikan **bobot attention**, terkait translation.

Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio, **Neural Machine Translation by Jointly Learning to Align and Translate, arXiv:1409.0473,** *2016*

Attention Mechanism

Simple Attention Networks untuk *Sentence Classification*



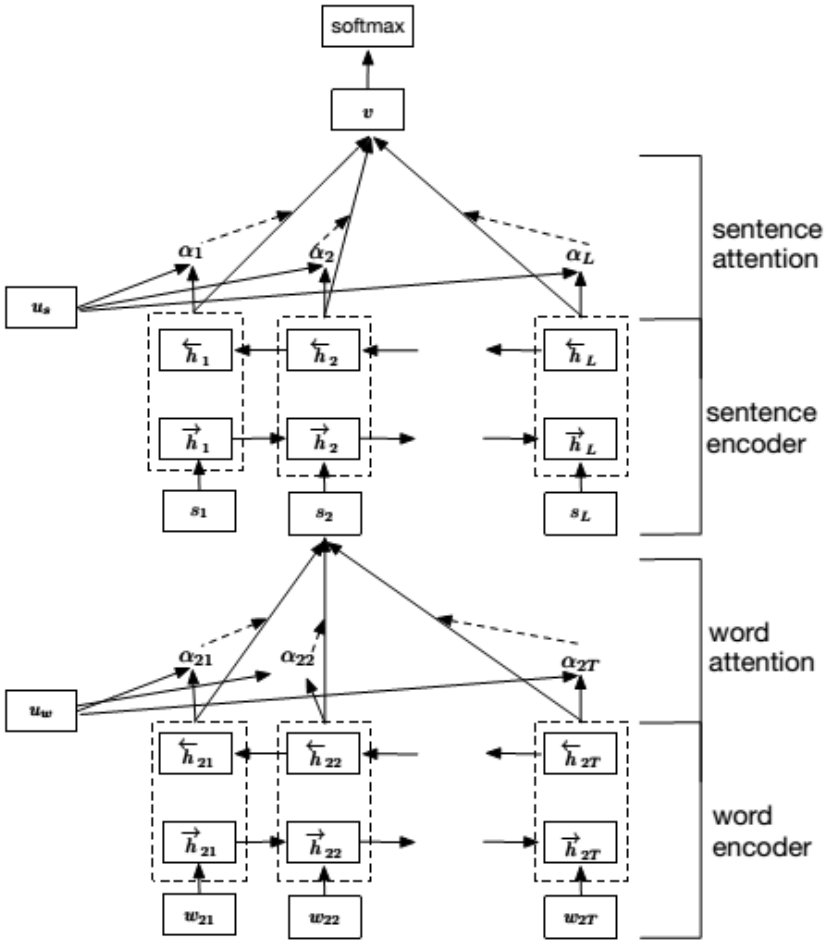
A straightforward simplification to the attention mechanism described above which would allow it to be used to produce a single vector c from an entire sequence could be formulated as follows:

$$e_t = a(h_t), \alpha_t = \frac{\exp(e_t)}{\sum_{k=1}^{T} \exp(e_k)}, c = \sum_{t=1}^{T} \alpha_t h_t \qquad (1)$$

Colin Raffel, Daniel P. W. Ellis, F EED -F ORWARD NETWORKS WITH ATTENTION CAN S OLVE S OME L ONG-T ERM MEMORY P ROBLEMS, Workshop track - ICLR 2016

Attention Mechanism

Hierarchical Attention Networks untuk *Sentence Classification*



Yang, Zichao, et al., Hierarchical Attention Networks for Document Classification, NAACL 2016

Attention Mechanism

Hierarchical Attention Networks untuk *Sentence Classification*

Task: Prediksi Rating Dokumen

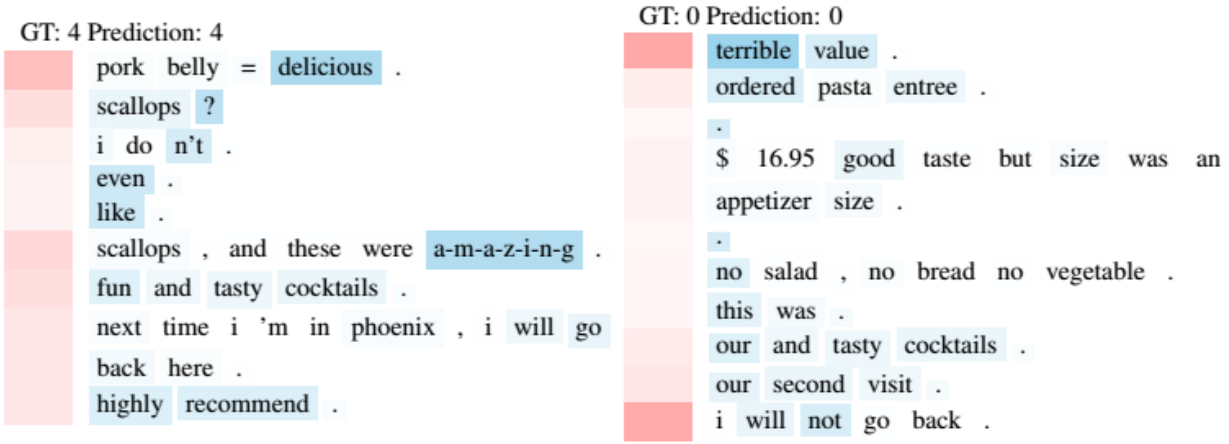


Figure 5: Documents from Yelp 2013. Label 4 means star 5, label 0 means star 1.

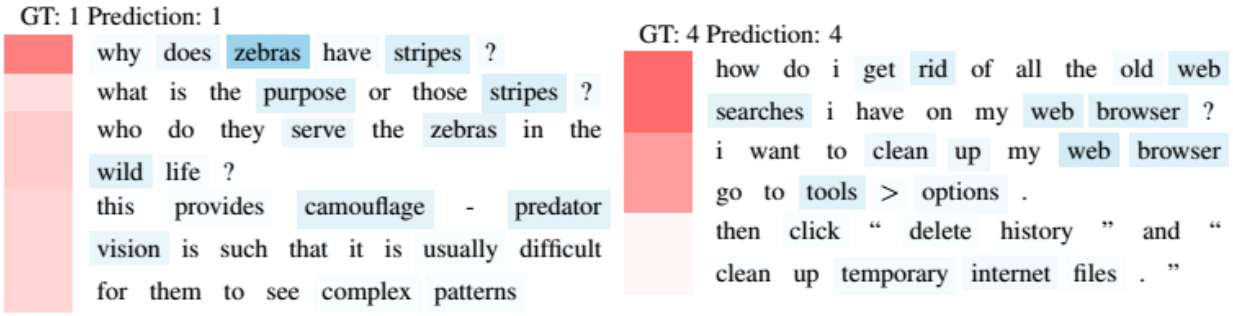
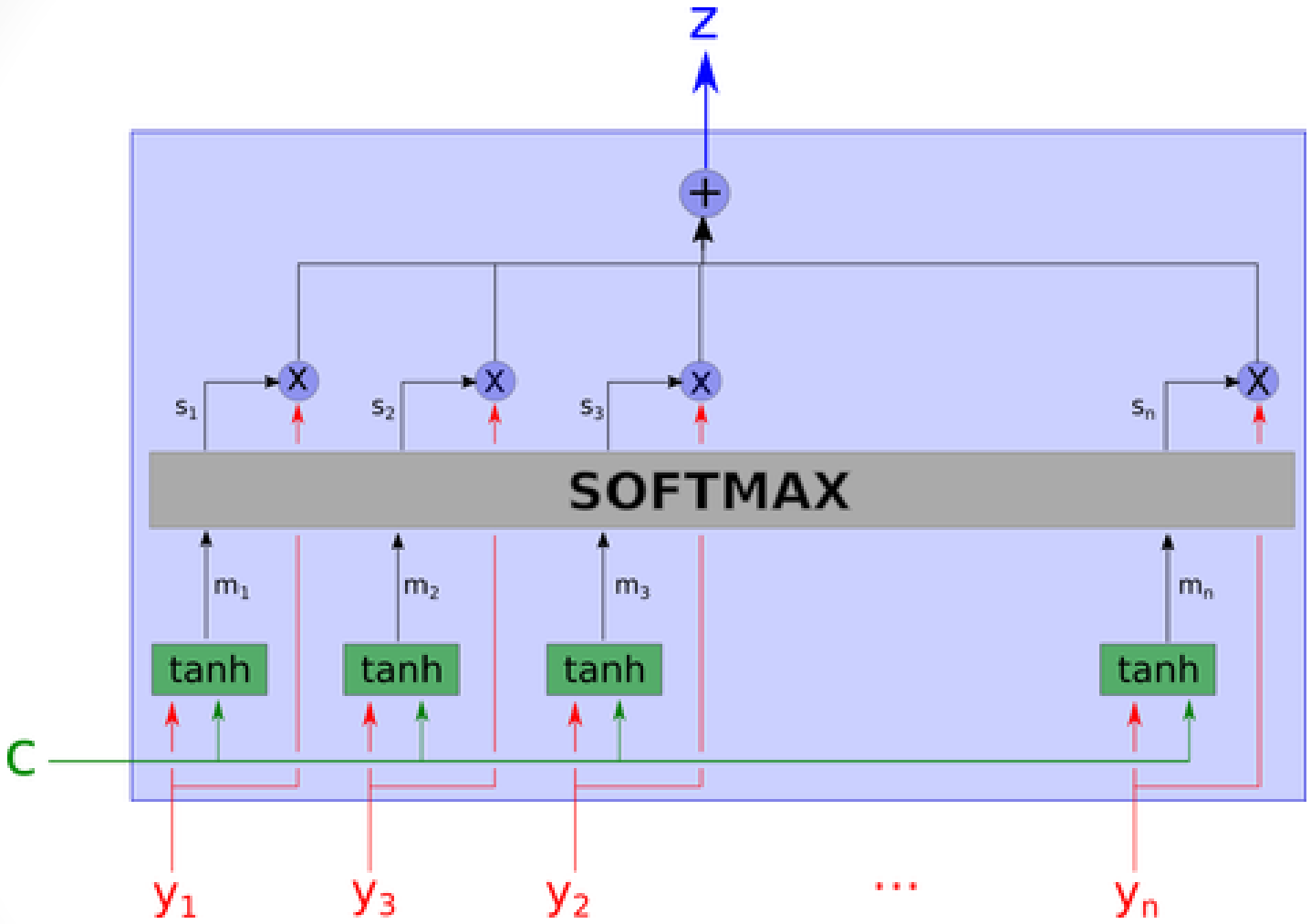


Figure 6: Documents from Yahoo Answers. Label 1 denotes Science and Mathematics and label 4 denotes Computers and Internet.

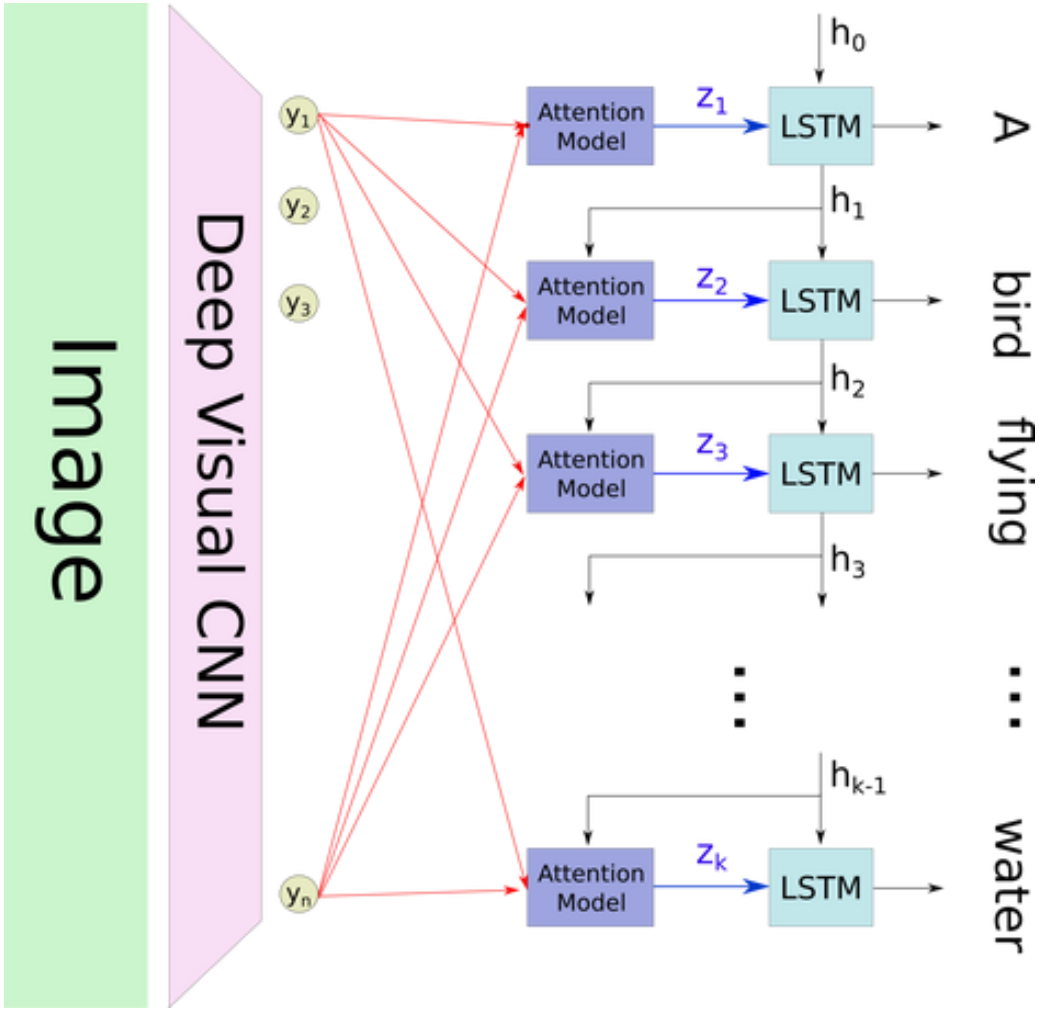
Yang, Zichao, et al., Hierarchical Attention Networks for Document Classification, NAACL 2016

Attention Mechanism



https://blog.heuritech.com/2016/01/20/attention-mechanism/

Attention Mechanism

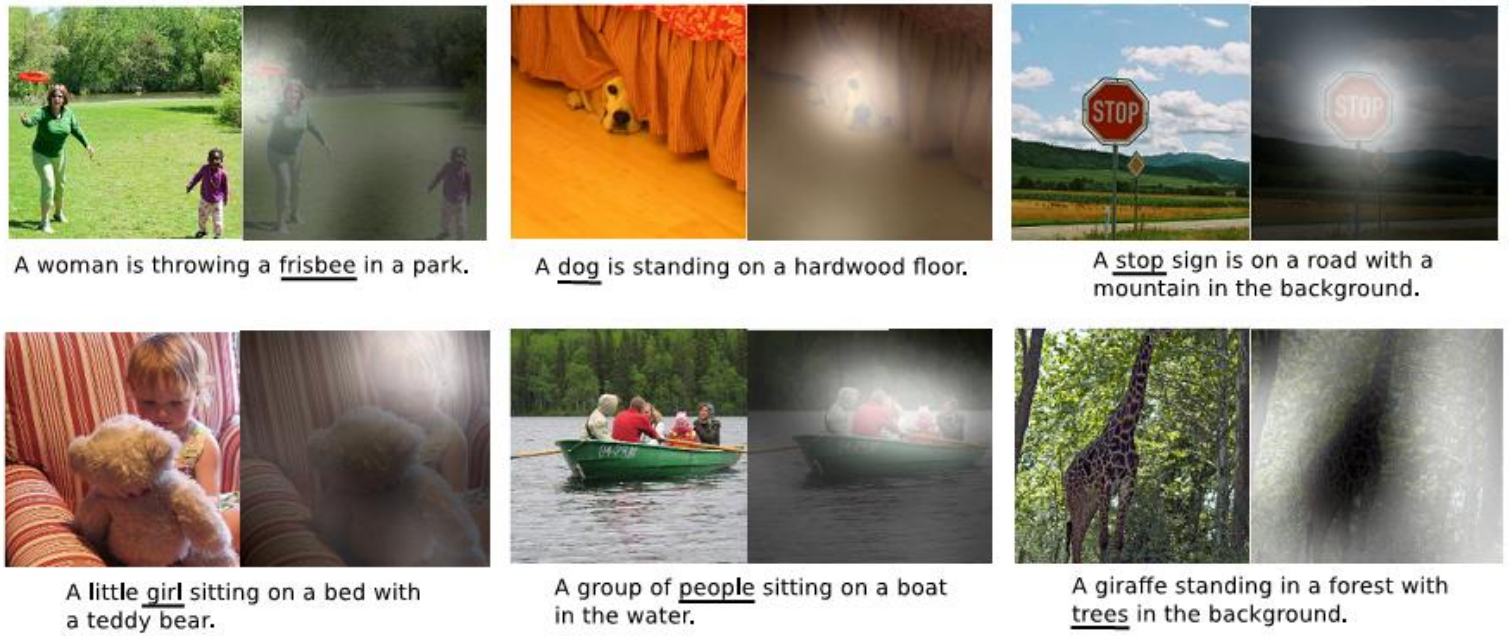


https://blog.heuritech.com/2016/01/20/attention-mechanism/

Xu, Kelvin, et al. « Show, Attend and Tell: Neural Image Caption Generation with Visual Attention (2016).

Attention Mechanism

Figure 3. Examples of attending to the correct object (*white* indicates the attended regions, *underlines* indicated the corresponding word)



https://blog.heuritech.com/2016/01/20/attention-mechanism/

Xu, Kelvin, et al. « Show, Attend and Tell: Neural Image Caption Generation with Visual Attention (2016).

Attention Mechanism

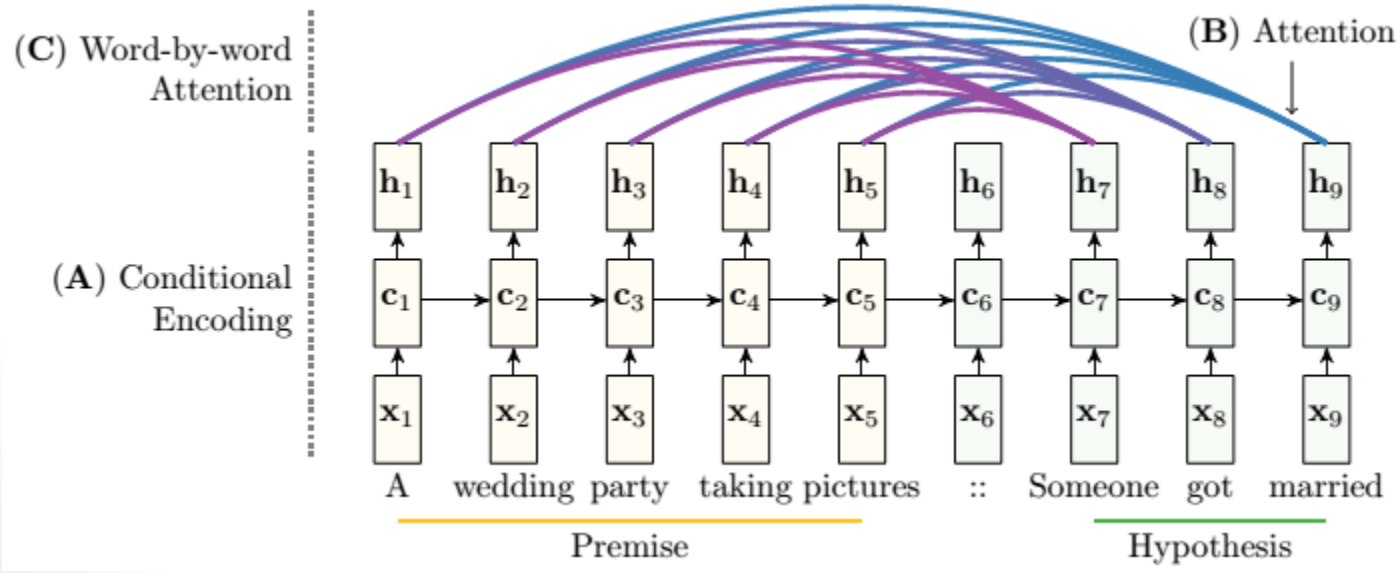
Attention Mechanism untuk Textual Entailment

Diberikan pasangan premis-hipotesis, tentukan apakah 2 tersebut **kontradiksi**, **tidak berhubungan**, atau **logically entail**.

Sebagai contoh:

- Premis: "*A wedding party taking pictures*"
- Hipotesis: "*Someone got married*"

Attention Model digunkan untuk menghubungkan kata-kata di premis dan hipotesis.

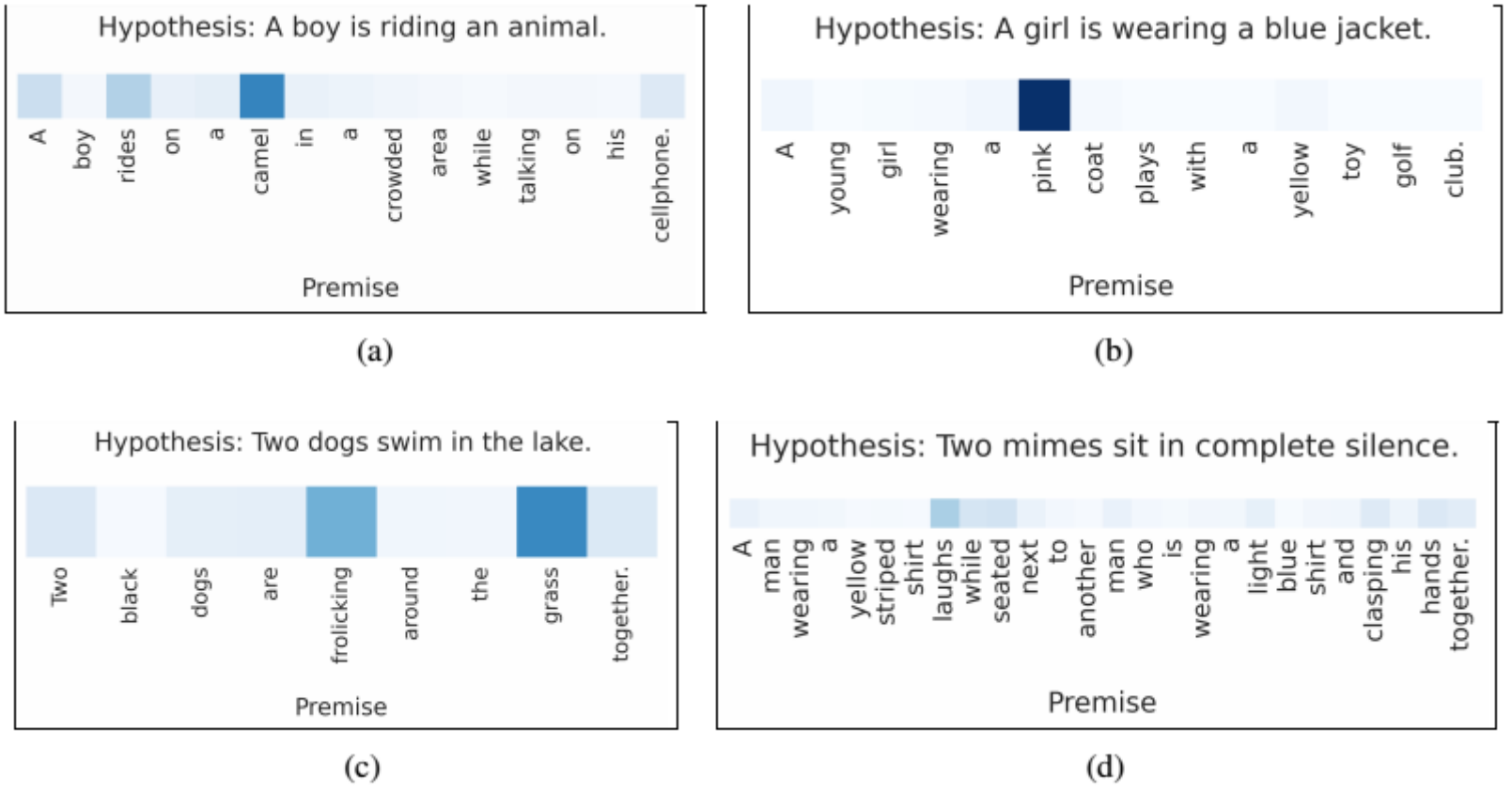


Tim Rocktaschel et al., REASONING ABOUT ENTAILMENT WITH NEURAL ATTENTION, ICLR 2016

Attention Mechanism

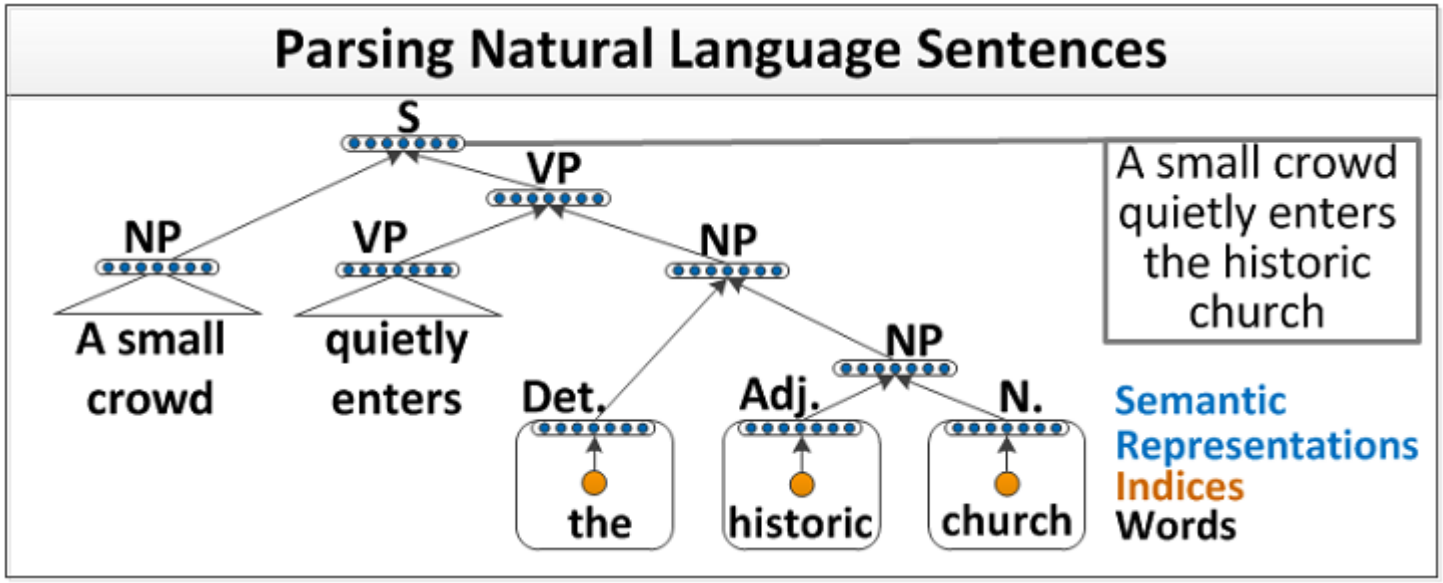
Attention Mechanism untuk Textual Entailment

Diberikan pasangan premis-hipotesis, tentukan apakah 2 tersebut **kontradiksi**, **tidak berhubungan**, atau **logically entail**.



Tim Rocktaschel et al., REASONING ABOUT ENTAILMENT WITH NEURAL ATTENTION, ICLR 2016

Recursive Neural Networks



R. Socher, C. Lin, A. Y. Ng, and C.D. Manning. 2011a. Parsing Natural Scenes and Natural Language with Recursive Neural Networks. In ICML

Recursive Neural Networks

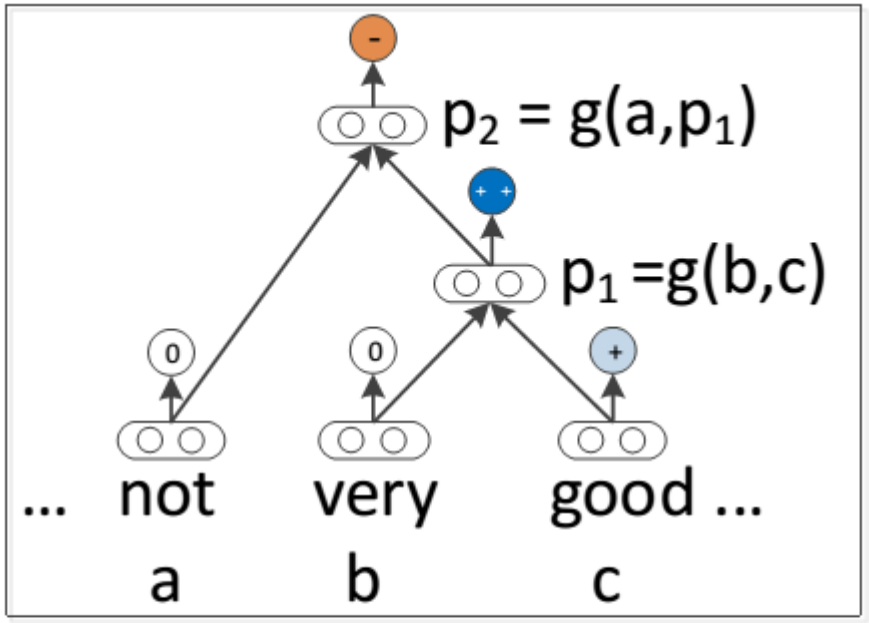
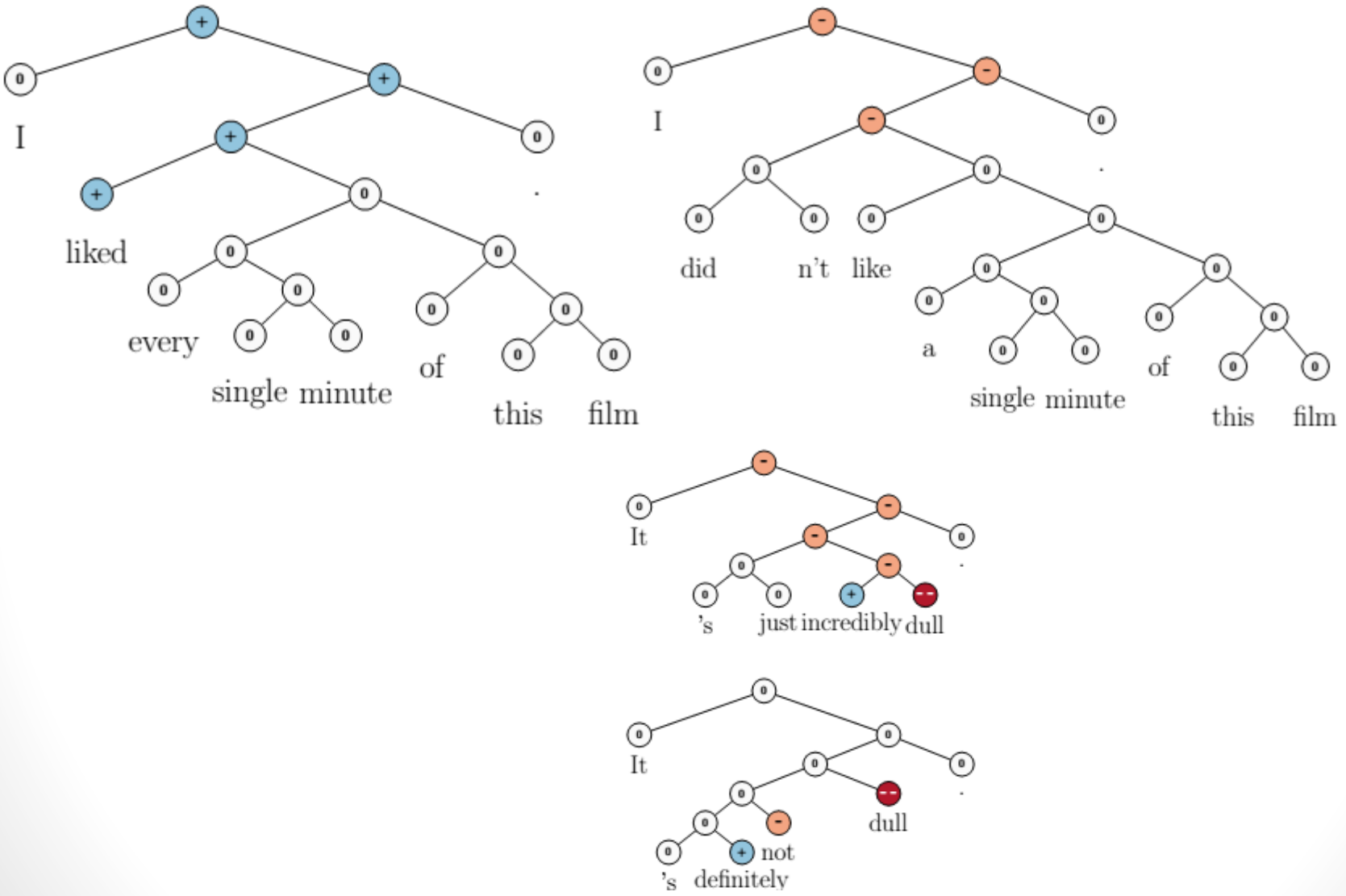


Figure 4: Approach of Recursive Neural Network models for sentiment: Compute parent vectors in a bottom up fashion using a compositionality function g and use node vectors as features for a classifier at that node. This function varies for the different models.

$$p_1 = g(W.[b;c] + bias)$$

Socher et al., Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank, EMNLP 2013

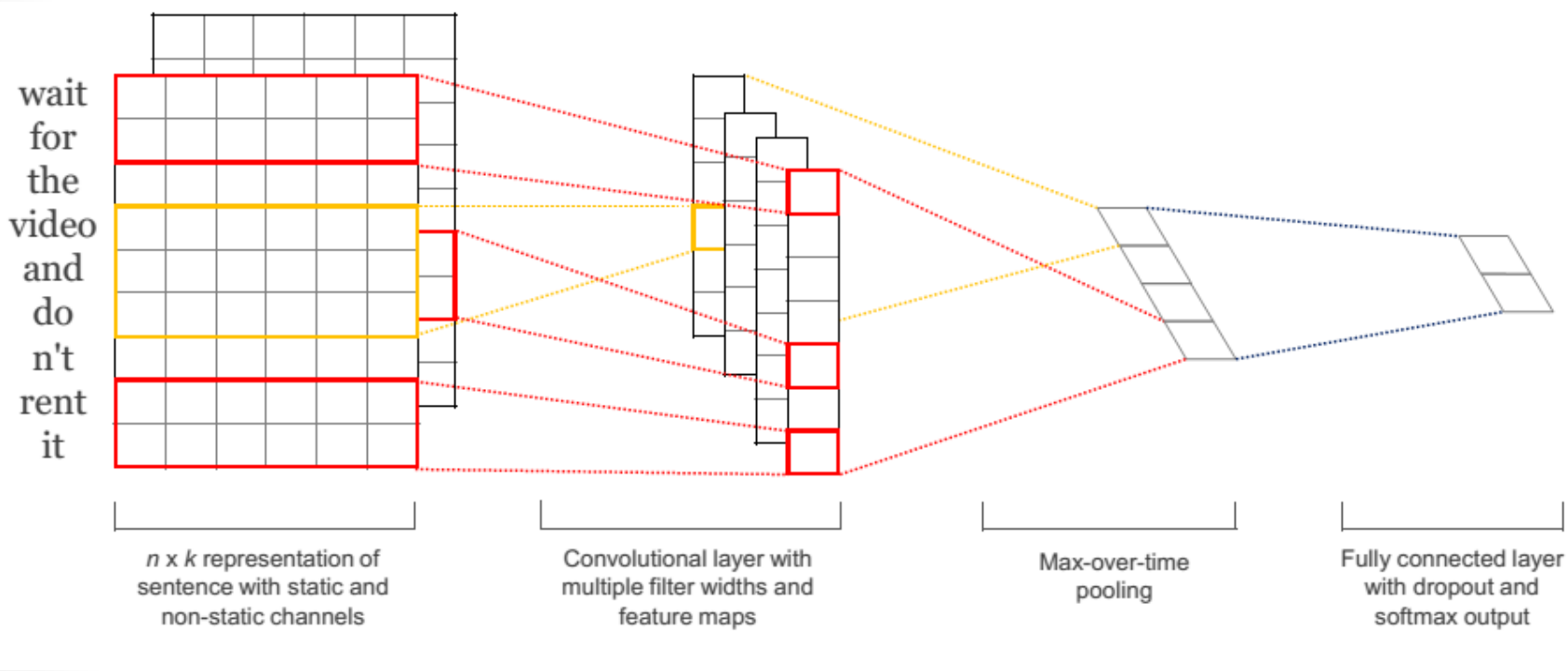
Recursive Neural Networks



Socher et al., Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank, EMNLP 2013

Convolutional Neural Networks (CNNs) for Sentence Classification

(Kim, EMNLP 2014)



Recursive Neural Network for SMT Decoding.

(Liu et al., EMNLP 2014)