

Content



0. Introduction

1. Regression

1.1 Multivariate Linear Regression (curve fitting)

1.2 Regularization (Lagrange multiplier)

1.3 Logistic Regression (Fermi-Dirac distribution)

1.4 Support Vector Machine (high-school geometry)

2. Dimensionality Reduction/feature extraction

2.1 Principal Component Analysis (order parameters)

2.2 Recommender Systems

2.3 Clustering (phase transition)

Content



3. Neural Networks

3.1 Biological neural networks

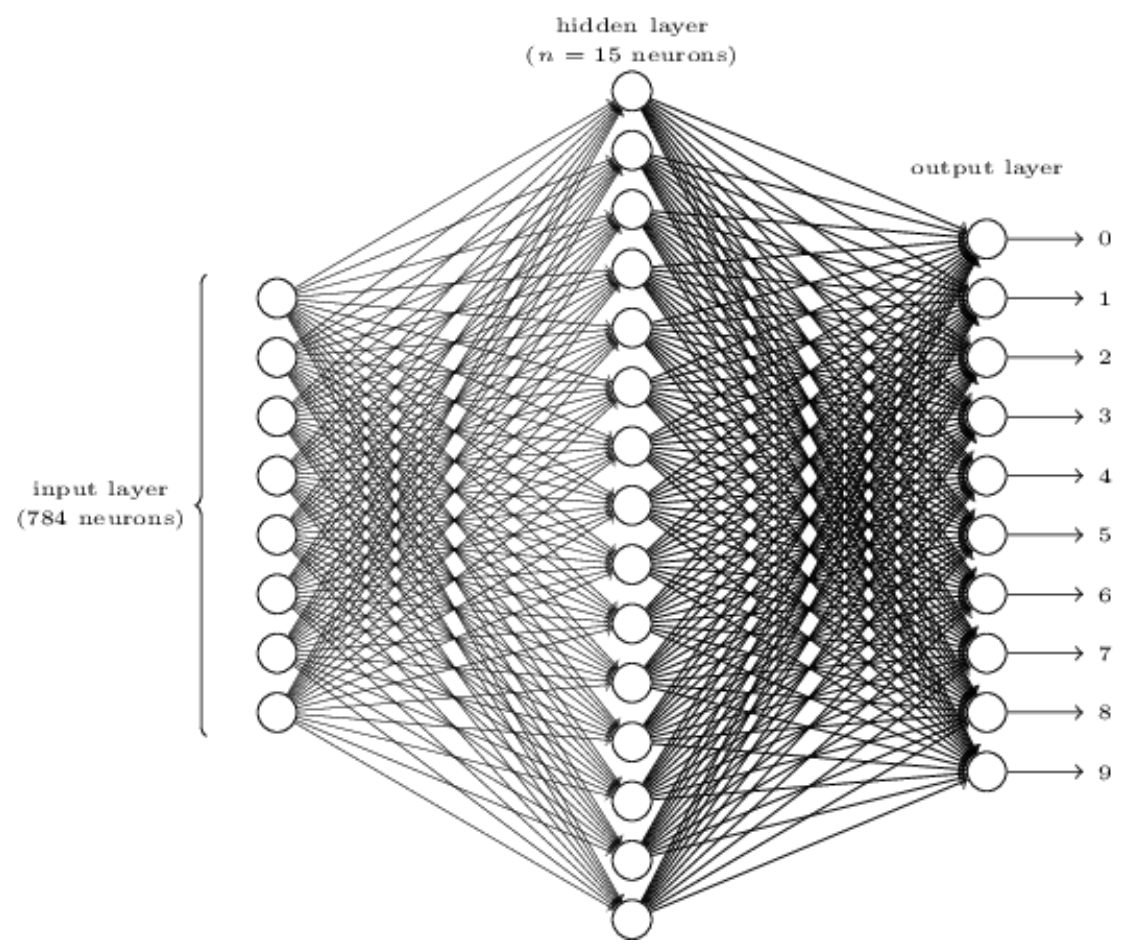
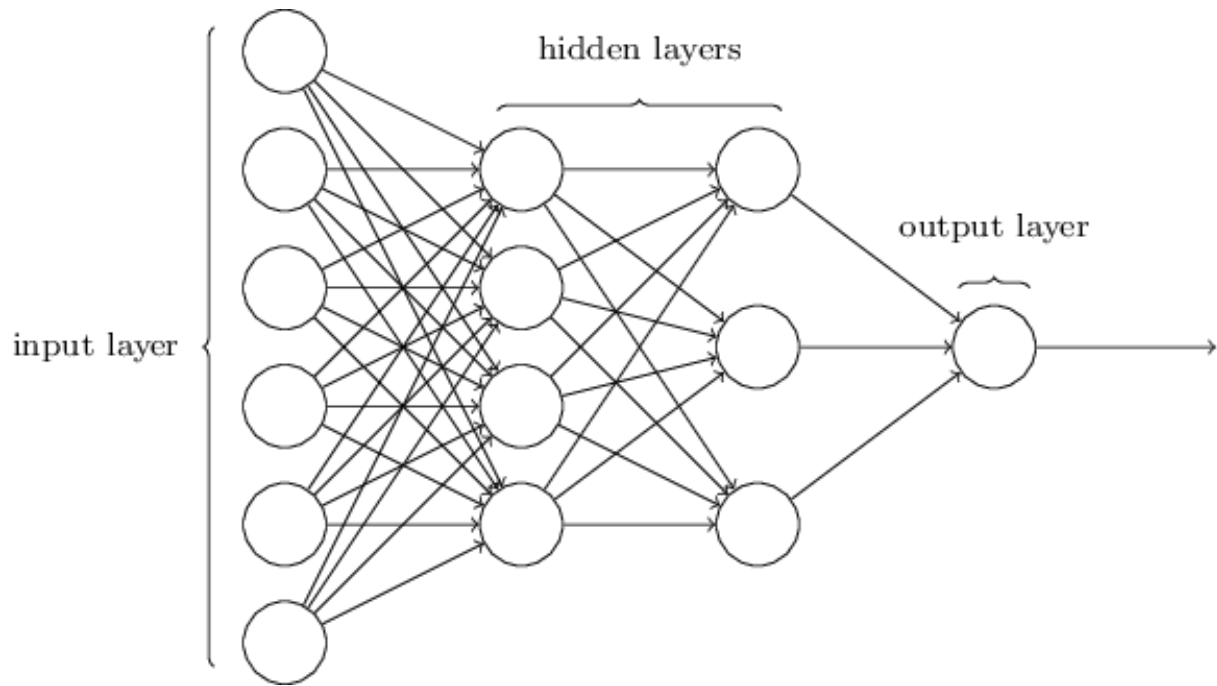
3.2 Mathematical representation

3.3 Feed-forward neural networks

3.5 Different Learning algorithm

3.6 Deep learning and CNN

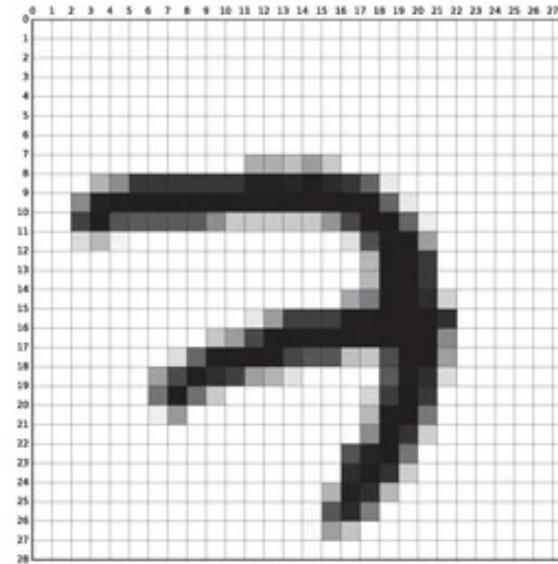
Four-layer network with two hidden layers



MNIST database of handwritten digits

Modified national institute of standards and technology database

60,000 training images and 10,000 testing images



(a) MNIST sample belonging to the digit '7'.



(b) 100 samples from the MNIST training set.

encode the intensities of the image pixels into the input neurons. If the image is a 28x28 greyscale image, then we'd have $28 \times 28 = 784$ input neurons, with the intensities scaled appropriately between 0 and 1.

Learning representations by back-propagating errors

[David E. Rumelhart](#), [Geoffrey E. Hinton](#) & [Ronald J. Williams](#)

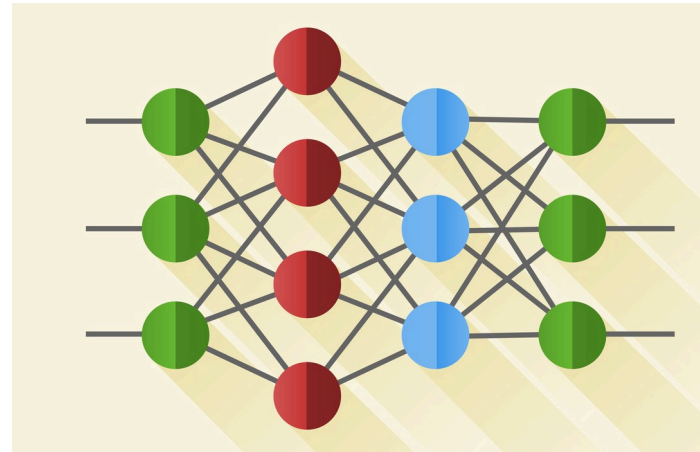
[Nature](#) 323, 533–536 (1986) | [Cite this article](#)

84k Accesses | 12307 Citations | 239 Altmetric | [Metrics](#)

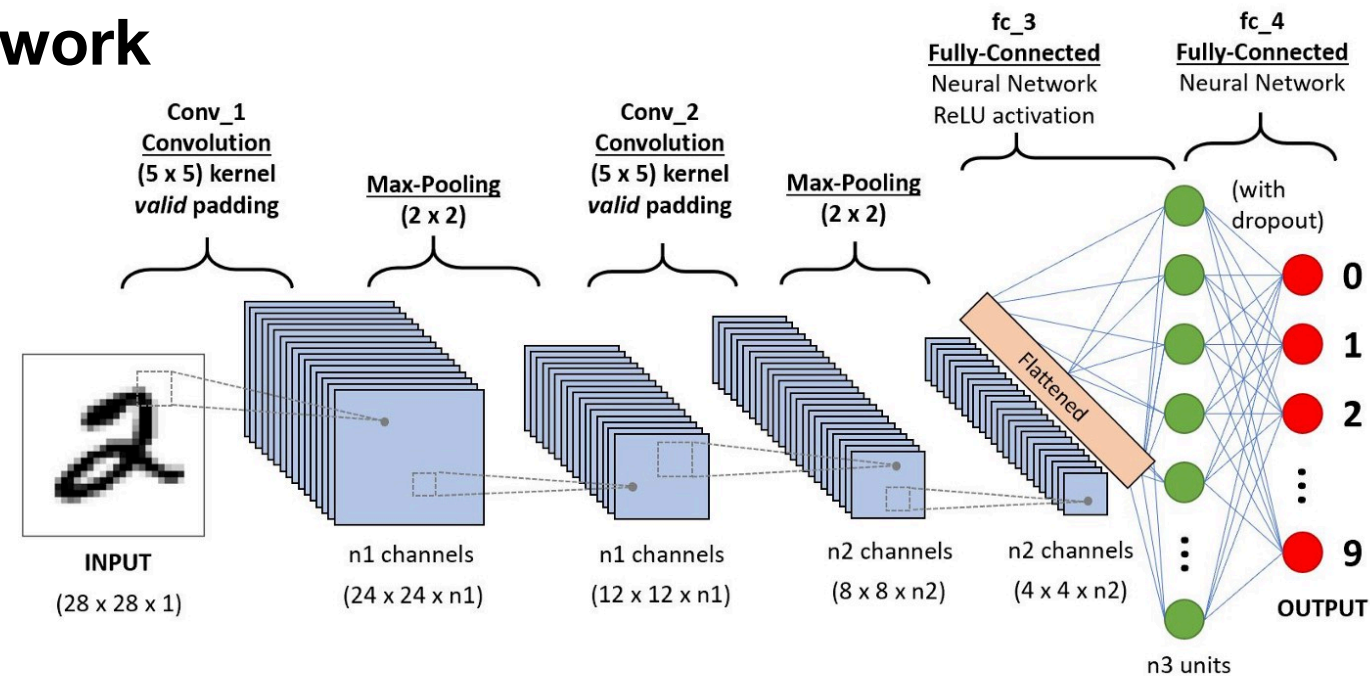
Connecting neurons to networks

Deep Learning: a network with 2 or more hidden layers with each layers containing large amount of neurons

Feed-forward network



Convolutional neural network



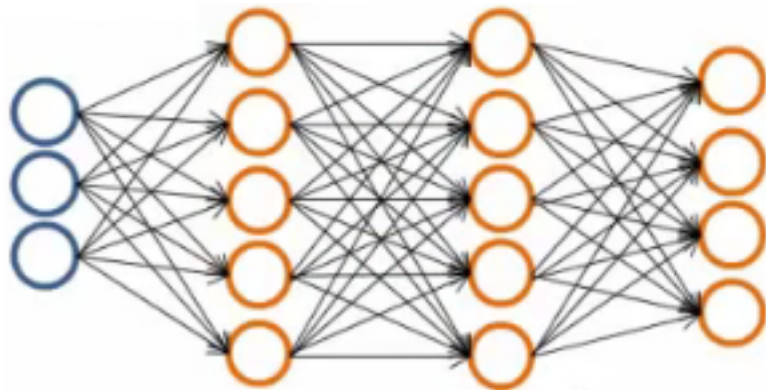
Generative neural network

Spiking neural network

<https://towardsdatascience.com/introduction-to-math-behind-neural-networks-e8b60dbbdeba>

<https://towardsdatascience.com/nns-aynk-c34efe37f15a>

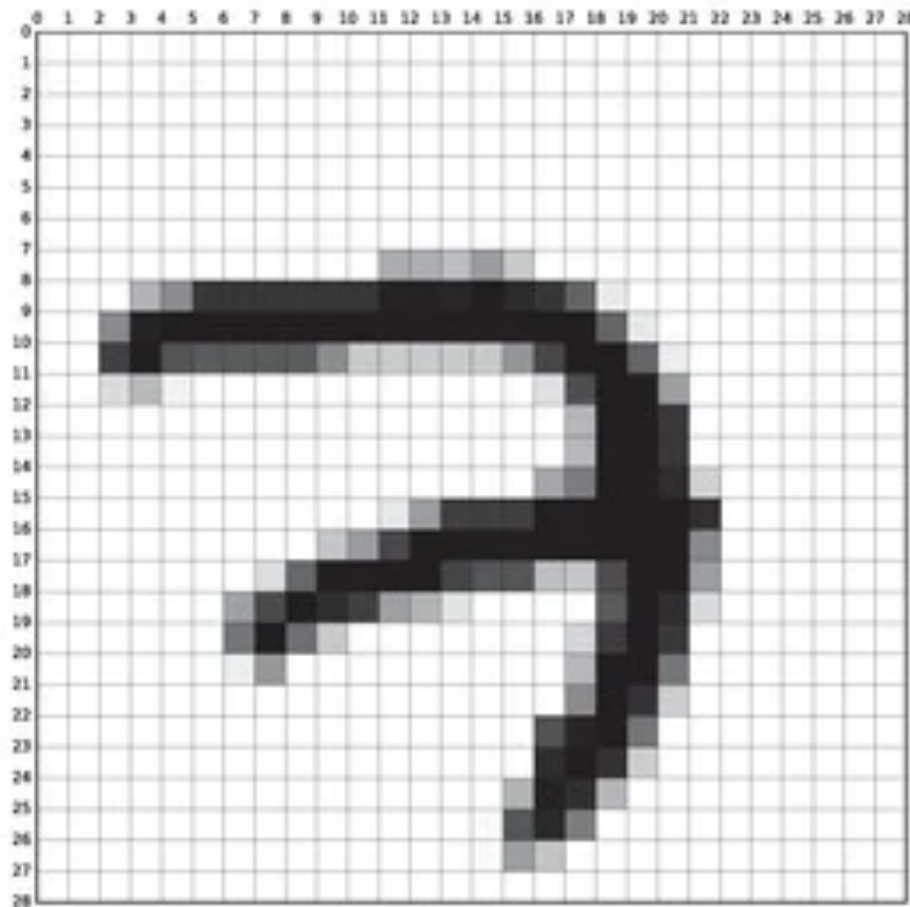
Neural network – Representation



Multiclass classification

$$h_{\Theta}(x) \in \mathbb{R}^4$$

Handwritten digital recognition problem - 10 possible categories (0-9)



(a) MNIST sample belonging to the digit '7'.



(b) 100 samples from the MNIST training set.

Mathematical representation for neurons: Learning Algorithm

Backpropagation: backward propagation of errors

Computing the gradient of the cost function with respect to the weights

Cost function $J = \text{mean squared error} = \frac{1}{m} \sum_{i=1}^m (y_i - \hat{y}_i)^2$

$$\frac{\partial J}{\partial w_i} = \frac{\partial J}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z} \times \frac{\partial z}{\partial w_i}$$

$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z} \times \frac{\partial z}{\partial b}$$

$\frac{2}{m} \sum_{i=1}^m (y_i - \hat{y}_i)$

$g(z)g(-z)$

x

1

$$z = w \cdot x + b = \theta^T x$$

$$y = \frac{1}{1 + \exp(-\theta^T x)}$$

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

1. $g(z) + g(-z) = 1$
2. $g'(z) = g(z)(1 - g(z)) = g(z)g(-z)$
3. $g'(-z) = g'(z)$

Optimization: select best weights and bias for the perceptron

$$w_i = w_i - \alpha \frac{\partial J}{\partial w_i} \qquad b = b - \alpha \frac{\partial J}{\partial b}$$

Andrew Ng, Stanford University

<http://www.holehouse.org/mlclass/>

<https://github.com/mxc19912008/Andrew-Ng-Machine-Learning-Notes>

Michael Nielsen, scientist at home, the best reading material

<http://neuralnetworksanddeeplearning.com>

<http://neuralnetworksanddeeplearning.com/chap1.html>

<http://neuralnetworksanddeeplearning.com/chap2.html>

<https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/>

Learning representations by back-propagating errors

[David E. Rumelhart](#), [Geoffrey E. Hinton](#) & [Ronald J. Williams](#)

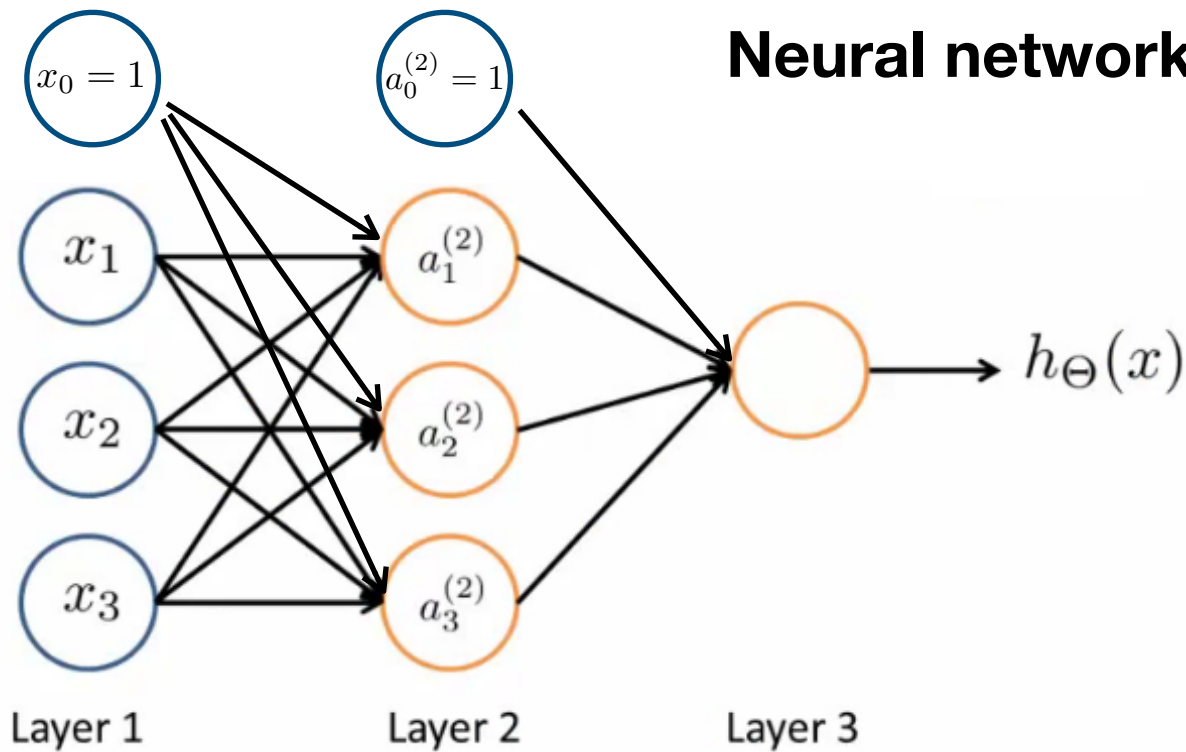
Nature **323**, 533–536 (1986) | [Cite this article](#)

84k Accesses | **12307** Citations | **239** Altmetric | [Metrics](#)

Early paper by
Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, Rob Fergus (2013)

<http://yann.lecun.com/exdb/publis/pdf/wan-icml-13.pdf>

Neural network – Representation



$a_i^{(l)}$ – activation of unit i in layer l

$\Theta^{(l)} : [s^{(l+1)} \times (s^{(l)} + 1)]$
 ← the bias

Matrix of parameters mapping from layer l to layer $l+1$

Weights

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$

$\Theta^{(1)} : [3 \times 4]$

$$h_{\Theta}(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}) \quad \Theta^{(2)} : [1 \times 4]$$

$$(z_1^2 = \theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3, z_2^2, z_3^2)$$

Vector representation

$$z^2 = \Theta^{(1)} x \quad a^2 = g(z^2)$$

$$z^3 = \Theta^{(2)} a^2 \quad h_{\Theta}(x) = a^3 = g(z^3)$$

Neural networks learns its own features

- Features in the hidden layer are calculated/learned - not original features
- Flexibility to learn whatever features it wants to feed into the final logistic regression calculation
- Hidden layers do the job, learn what gives the best final results to feed into final output layer
- Non-linear hypothesis

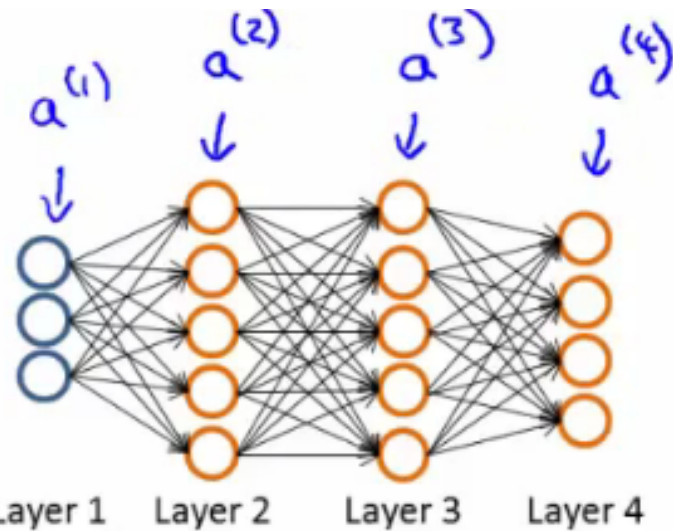
Neural network – Learning – back-propagation

Learning representations by back-propagating errors

[David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams](#)

[Nature](#) 323, 533–536 (1986) | [Cite this article](#)

84k Accesses | 12307 Citations | 239 Altmetric | [Metrics](#)



$$L = 4 \quad s^1 = 3 \quad s^2 = 5 \quad s^3 = 5 \quad s^4 = 4$$

Training set (in vector form)

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$$

Forward propagation

$$a^1 = x$$

$$z^2 = \Theta^1 a^1 \text{ (add } x_0)$$

$$a^2 = g(z^2)$$

$$z^3 = \Theta^2 a^2 \text{ (add } a_0^2)$$

$$a^3 = g(z^3)$$

$$z^4 = \Theta^3 a^3 \text{ (add } a_0^3)$$

$$a^4 = h_{\Theta}(x) = g(z^4)$$

Back propagation

δ_j^l – the error of neuron j in layer l

$$\delta^1 = (\Theta^1)^T \delta^2 - x$$

$\cdot *$ elementwise multiplication
Hadamard product

$$\delta^2 = (\Theta^2)^T \delta^3 \cdot * g'(z^2)$$

$$g'(z) = g(z) \cdot * (1 - g(z))$$

$$\delta^3 = (\Theta^3)^T \delta^4 \cdot * g'(z^3)$$

$$\Theta^3 : [4 \times 5] \text{ } ([4 \times 6] \text{ if include bias)}$$

$$(\Theta^3)^T : [5 \times 4]$$

$$\delta^4 = (a^4 - y) \cdot * g'(z^4)$$

$$\delta^4 : [4 \times 1]$$

$$g'(z^3) : [5 \times 1]$$

$$\tilde{\delta}_j^4 = a_j^4 - y_j$$

from quadratic cost

Neural network – Learning – back-propagation

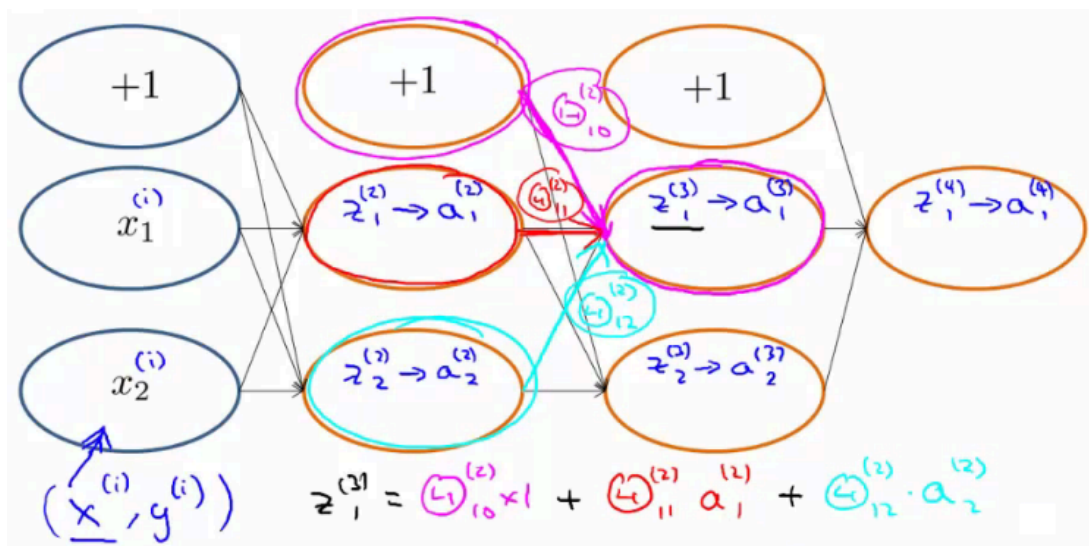
$$J(\Theta) = \frac{1}{2m} \sum_x ||y(x) - a^L(x)||^2 = \frac{1}{2m} \sum_i ||a_i^L - y_i||^2$$

Training set (in vector form) $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$

$$\frac{\partial J(\Theta)}{\partial \Theta_{ij}^{(l)}} = a_j^{(l)} \delta_i^{(l+1)} = a_{\text{in}} \delta_{\text{out}}$$

Loop through the training set $i = 1$ to m

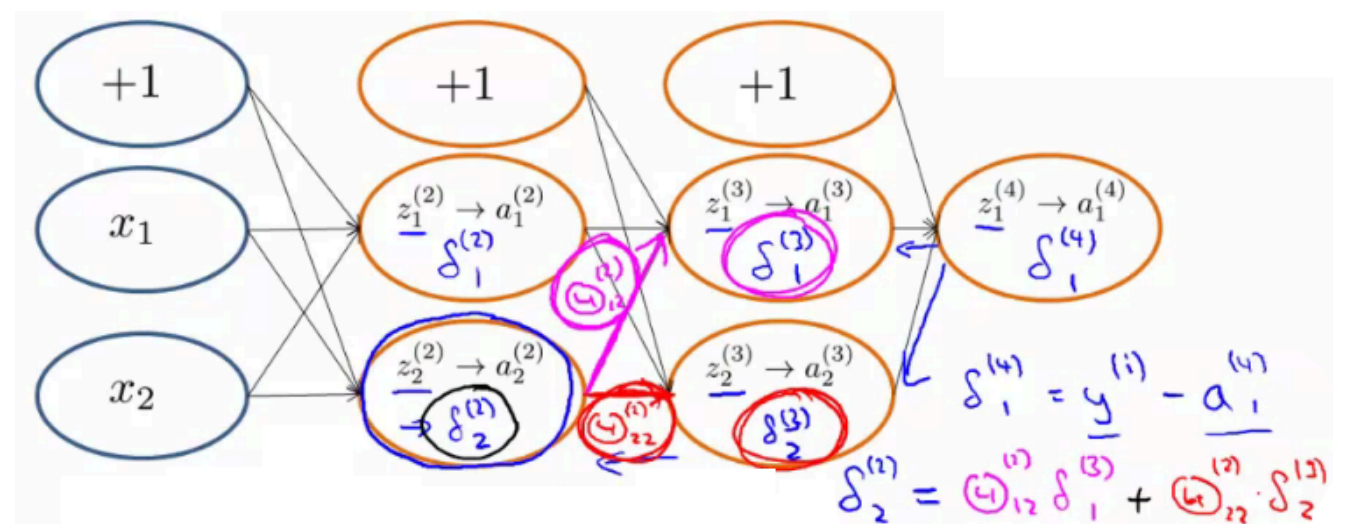
Forward propagation



Calculate activation values: weighted sum of the previous layer's activation values, weighted by the link parameters

Back propagation

How well is the network doing on example i ?



Calculate delta values: weighted sum of the next layer's delta values, weighted by the link parameters

After executing the loop

Update the cost function

$$\Theta_{ij}^{(l)} = \Theta_{ij}^{(l)} - \alpha \frac{\partial J(\Theta)}{\partial \Theta_{ij}^{(l)}}$$

$$\frac{\partial J}{\partial \Theta_{10}^{(1)}} = \delta_1^{(2)} \quad \frac{\partial J}{\partial \Theta_{10}^{(2)}} = \delta_1^{(3)}$$

$$\frac{\partial J}{\partial \Theta_{12}^{(1)}} = a_2^{(1)} \delta_1^{(2)} \quad \frac{\partial J}{\partial \Theta_{12}^{(2)}} = a_2^{(2)} \delta_1^{(3)}$$

$$\frac{\partial J}{\partial \Theta_{22}^{(1)}} = a_2^{(1)} \delta_2^{(2)} \quad \frac{\partial J}{\partial \Theta_{22}^{(2)}} = a_2^{(2)} \delta_2^{(3)}$$

Neural network – Learning put together

1) - pick a network architecture

Input units (x , a_1)

Output units (y)

Hidden units (Θ_1 , a_2 , Θ_2 , a_3 ...)

Normally more hidden units is better but more computationally expensive

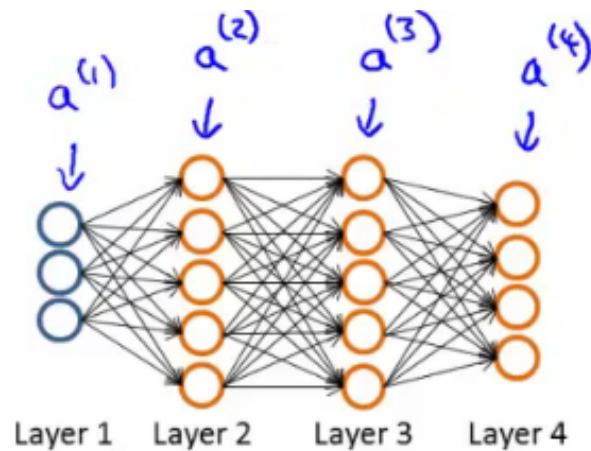
2) - training a network

Randomly initialize the weights

Implement forward propagation to get a_L for any example x_i

Implement cost function $J(\Theta)$

Implement back propagation to get partial derivatives



```
for i = 1:m {  
    Forward propagation on  $(x^i, y^i)$  --> get activation ( $a$ ) terms  
    Back propagation on  $(x^i, y^i)$  --> get delta ( $\delta$ ) terms  
    Compute  $\Delta := \Delta^1 + \delta^{l+1} (a^1)^T$   
}
```

With this done compute the partial derivative terms

Use (gradient descent, CG or more advanced optimization) to minimise $J(\Theta)$:

$J(\Theta)$ is non-convex

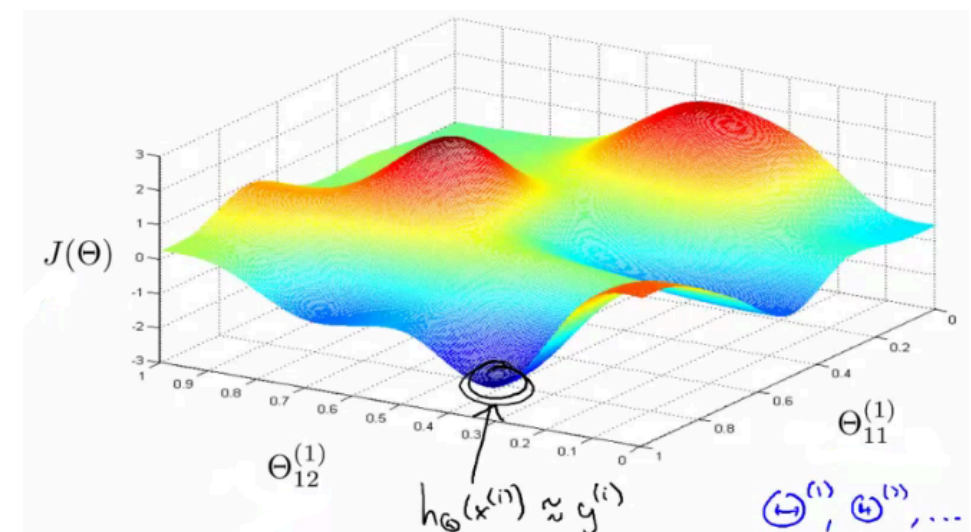
Nonlinearity, local minimum, time, complexity ...

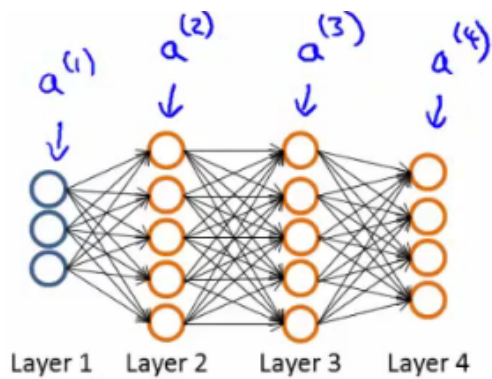
Learning representations by back-propagating errors

[David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams](#)

[Nature](#) **323**, 533–536 (1986) | [Cite this article](#)

84k Accesses | 12307 Citations | 239 Altmetric | [Metrics](#)





$$\delta_j^l = \frac{\partial J}{\partial z_j^l} \text{ measures the error of neuron } j \text{ in layer } l$$

An equation for the rate of change of the cost with respect to any weight in the network

Partial derivatives $\frac{\partial}{\partial \Theta_{ij}^l} J(\Theta) = a_j^l \delta_i^{l+1}$

Learn slowly if either the input neuron is low-activation, or if the output neuron has saturated, i.e., is either high- or low-activation.

Summary: the equations of backpropagation

$\delta^L = \nabla_a C \odot \sigma'(z^L)$ (BP1)

$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$ (BP2)

$\frac{\partial C}{\partial b_j^l} = \delta_j^l$ (BP3)

$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$ (BP4)

Prove this $\delta_j^2 = \frac{\partial J}{\partial z_j^2} = \sum_k \frac{\partial J}{\partial z_k^3} \frac{\partial z_k^3}{\partial z_j^2} = \sum_k \delta_k^3 \frac{\partial z_k^3}{\partial z_j^2}$

$z_k^3 = \sum_j \Theta_{kj}^2 a_j^2 = \sum_j \Theta_{kj}^2 g(z_j^2)$ $\frac{\partial z_k^3}{\partial z_j^2} = \Theta_{kj}^2 g'(z_j^2)$

$\delta_j^2 = \sum_k \Theta_{kj}^2 \delta_k^3 g'(z_j^2)$ $\delta^2 = (\Theta^2)^T \delta^3 \cdot * g'(z^2)$

Notice the shift of l by 1, index jungle

Prove this $\frac{\partial J}{\partial \Theta^2} = \frac{\partial J}{\partial z^3} \frac{\partial z^3}{\partial \Theta^2} = \delta^3 a^2 = a^2 \delta^3$

$\delta^3 = \frac{\partial J}{\partial z^3}$ $z^3 = \Theta^2 a^2$

In particular, given a mini-batch of m training examples, the following algorithm applies a gradient descent learning step based on that mini-batch

1. Input a set of training examples

2. For each training example x : Set the corresponding input activation $a^{x,1}$, and perform the following steps:

- **Feedforward:** For each $l = 2, 3, \dots, L$ compute

$$z^{x,l} = w^l a^{x,l-1} + b^l \text{ and } a^{x,l} = \sigma(z^{x,l}).$$

- **Output error $\delta^{x,L}$:** Compute the vector

$$\delta^{x,L} = \nabla_a C_x \odot \sigma'(z^{x,L}).$$

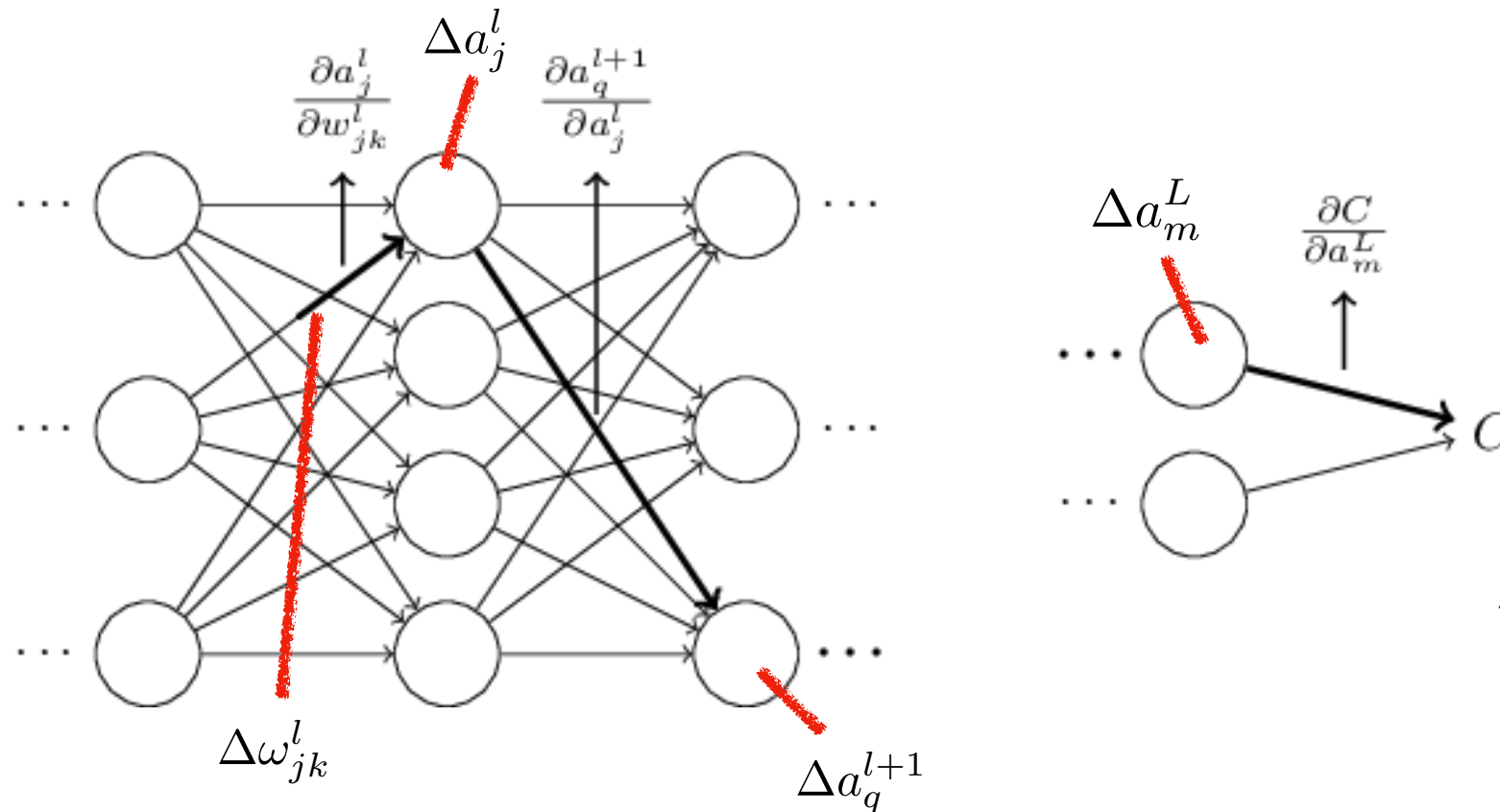
- **Backpropagate the error:** For each

$l = L - 1, L - 2, \dots, 2$ compute

$$\delta^{x,l} = ((w^{l+1})^T \delta^{x,l+1}) \odot \sigma'(z^{x,l}).$$

3. Gradient descent: For each $l = L, L - 1, \dots, 2$ update the weights according to the rule $w^l \rightarrow w^l - \frac{\eta}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T$, and the biases according to the rule $b^l \rightarrow b^l - \frac{\eta}{m} \sum_x \delta^{x,l}$.

Sigmoid is differentiable



$$\Delta C \approx \frac{\partial C}{\partial w_{jk}^l} \Delta w_{jk}^l.$$

$$\Delta a_j^l \approx \frac{\partial a_j^l}{\partial w_{jk}^l} \Delta w_{jk}^l \quad \Delta a_q^{l+1} \approx \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l} \Delta w_{jk}^l \quad \dots \quad \Delta C \approx \frac{\partial C}{\partial a_m^L} \frac{\partial a_m^L}{\partial a_n^{L-1}} \frac{\partial a_n^{L-1}}{\partial a_p^{L-2}} \dots \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l} \Delta w_{jk}^l$$

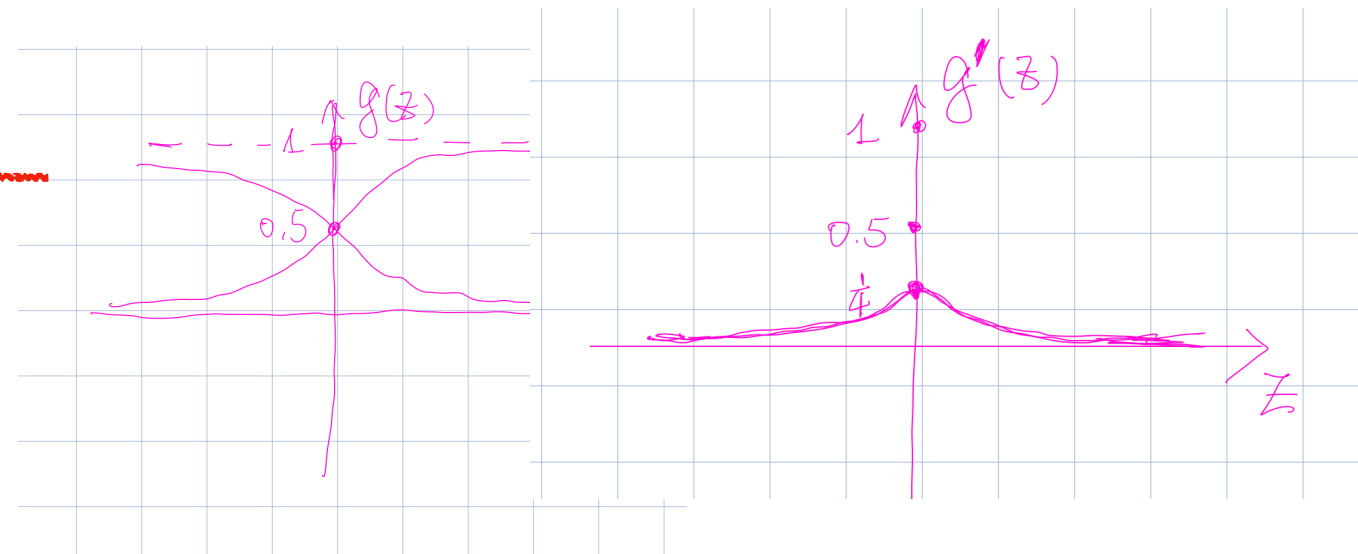
$$\Delta C \approx \sum_{mnp\dots q} \frac{\partial C}{\partial a_m^L} \frac{\partial a_m^L}{\partial a_n^{L-1}} \frac{\partial a_n^{L-1}}{\partial a_p^{L-2}} \dots \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l} \Delta w_{jk}^l$$

$$\frac{\partial C}{\partial w_{jk}^l} = \sum_{mnp\dots q} \frac{\partial C}{\partial a_m^L} \frac{\partial a_m^L}{\partial a_n^{L-1}} \frac{\partial a_n^{L-1}}{\partial a_p^{L-2}} \dots \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l}$$

<http://neuralnetworksanddeeplearning.com/chap2.html>

Improving the way neural networks learn

$$\frac{\partial}{\partial \Theta_{ij}^l} J(\Theta) = a_j^l \delta_i^{l+1} \quad \delta^3 = (\Theta^3)^T \delta^4 \cdot * g'(z^3)$$



Learning slowdown when output neuron saturates

Cross-entropy as cost function (cost function of logistic regression)

$$J(\Theta) = -\frac{1}{m} \sum_{i=1}^m [y_i \ln(a_i^L) + (1 - y_i) \ln(1 - a_i^L)]$$

$$\frac{\partial J}{\partial \theta_i} = \frac{1}{m} \sum_{i=1}^m x_i (g(z_i) - y_i)$$

Error in output control the learning

$$\frac{\partial J}{\partial b} = \frac{1}{m} \sum_{i=1}^m (g(z_i) - y_i)$$

Regularization

$$J(\Theta) = -\frac{1}{m} \sum_{i=1}^m [y_i \ln(a_i^L) + (1 - y_i) \ln(1 - a_i^L)] + \frac{\lambda}{2m} \sum_{\Theta} \Theta^2$$