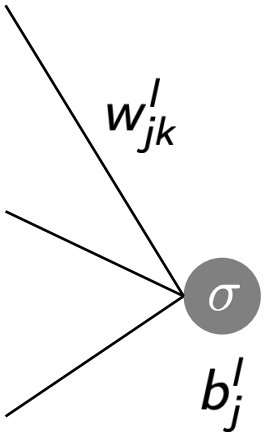
Computational MRI

Training neural networks with backpropagation

Neural networks



$$a^l_j = \sigma\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right)$$

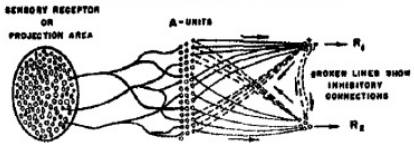


FIG. 2A. Schematic representation of connections in a simple perceptron.

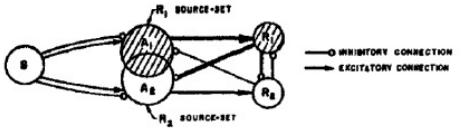
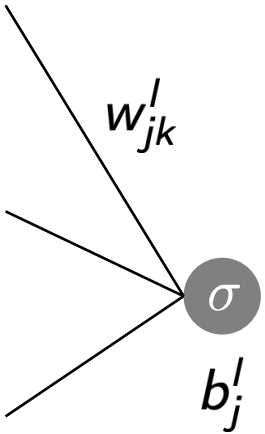


FIG. 2B. Venn diagram of the same perceptron (shading shows active sets for R_1 response).

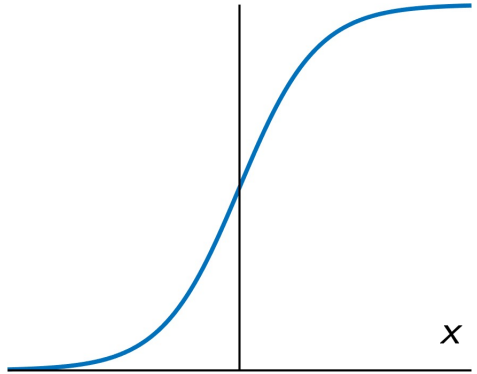
Rosenblatt Psychological review 1957

Neural networks



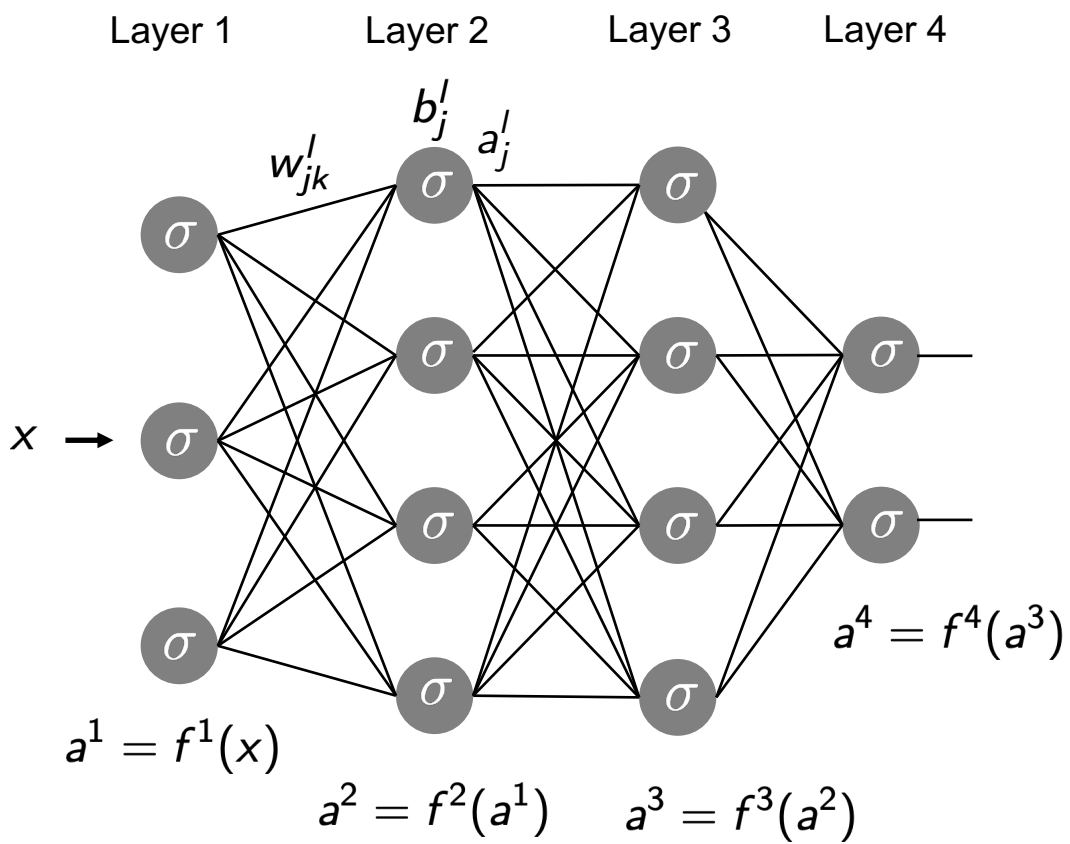
$$a^l_j = \sigma\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right)$$

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



Rosenblatt Psychological review 1957

Neural networks



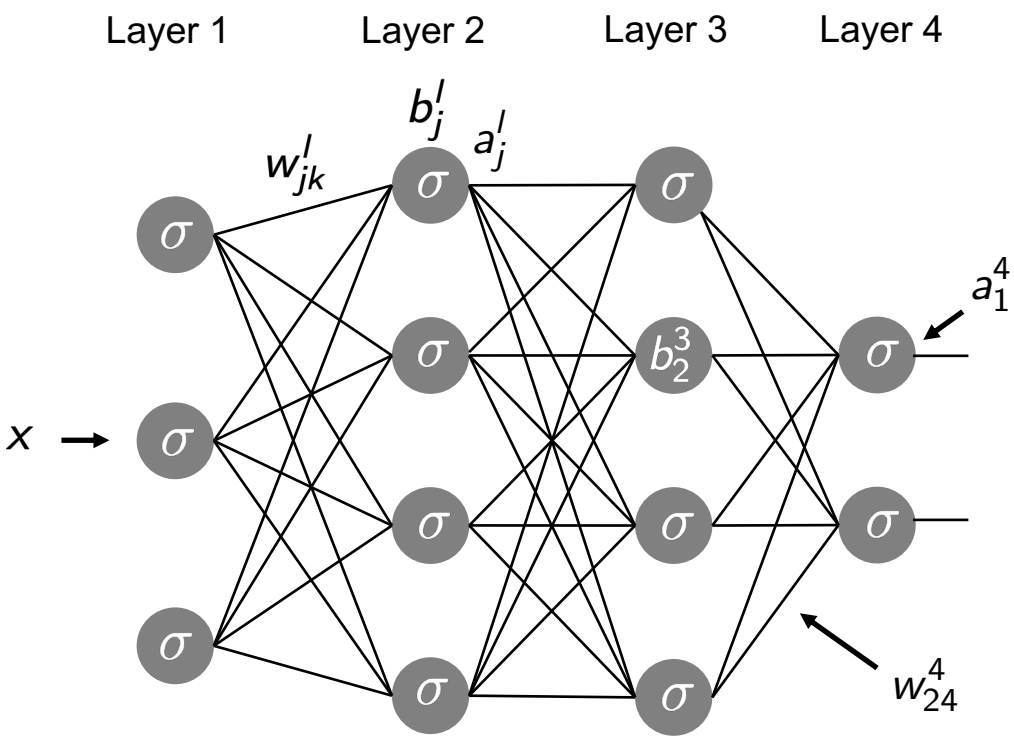
Large number of model parameters

High descriptive capacity

$$a^4 = F(x) = f^4(f^3(f^2(f^1(x))))$$

Cybenko: Math Control Sig Sys 1989

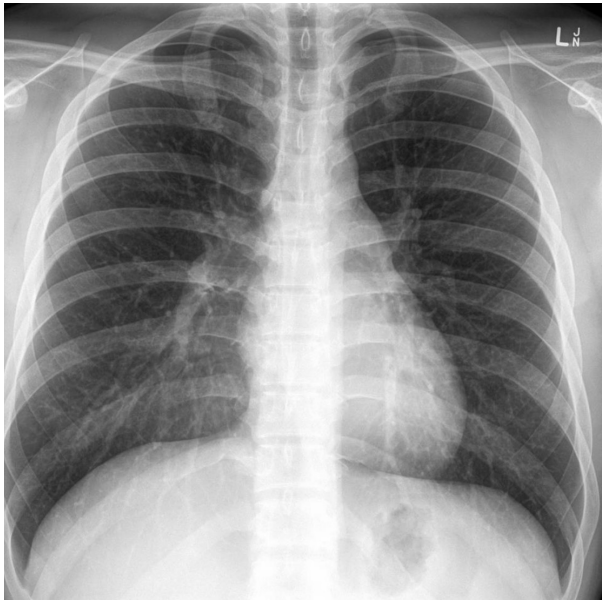
Neural networks: Notation



layer 4 which connects 2nd element and 4 element

ER chest X-Ray diagnosistic classification

Normal

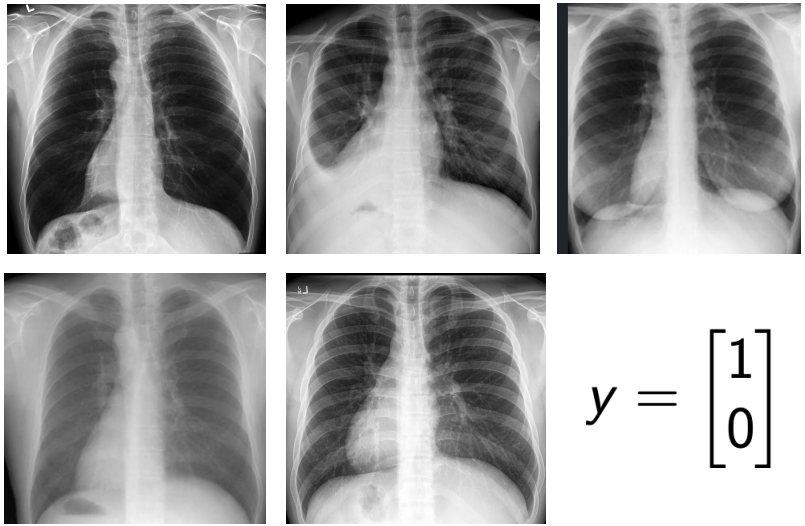


COVID-19

Radiopedia.org

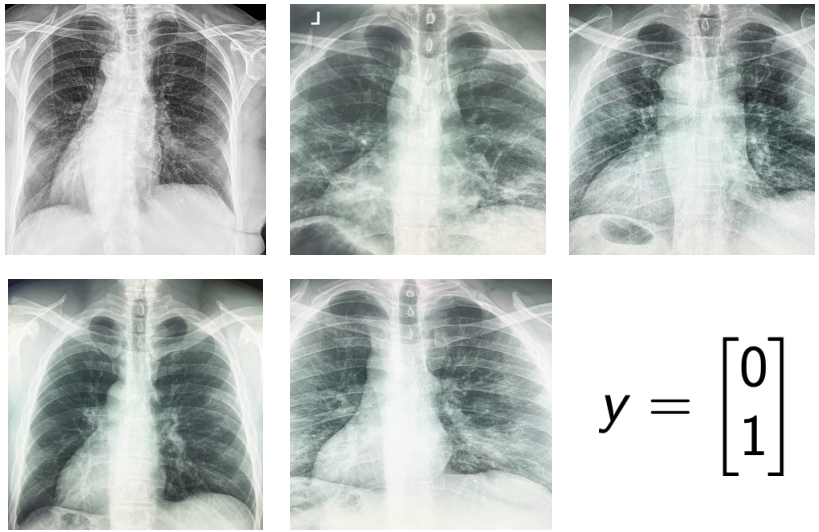
Chest X-Ray data set

Normal



$$y = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

COVID-19



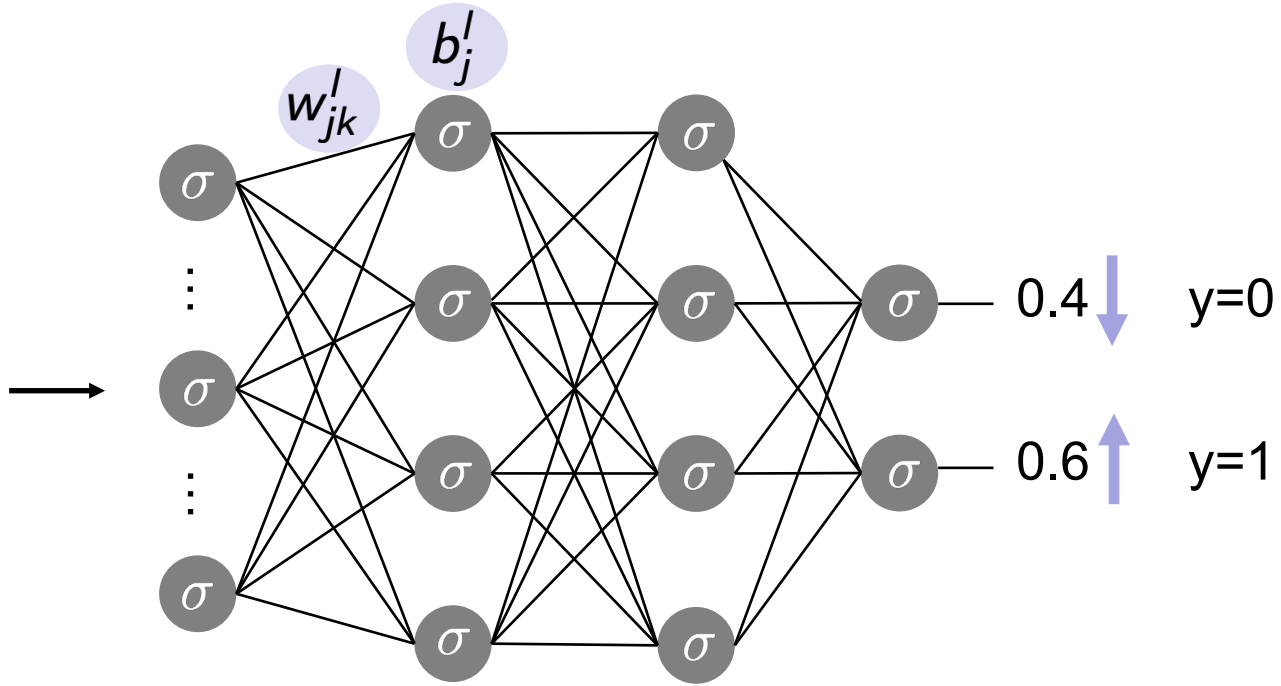
$$y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\{(x_1, y_1), \ldots (x_N, y_N)\}$$

Radiopedia.org
Weinstock JUCM 2020

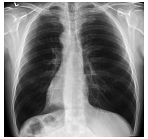
Neural network training

COVID-19

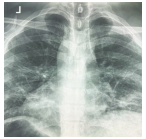


Change weights and biases to bring output closer to target

Neural network training: Cost function



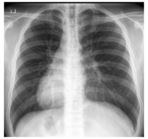
$y = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$



$y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

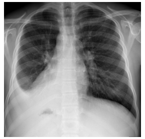
$y = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

$y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$



$y = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

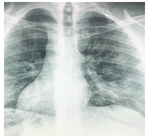
$y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$



$y = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

$y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

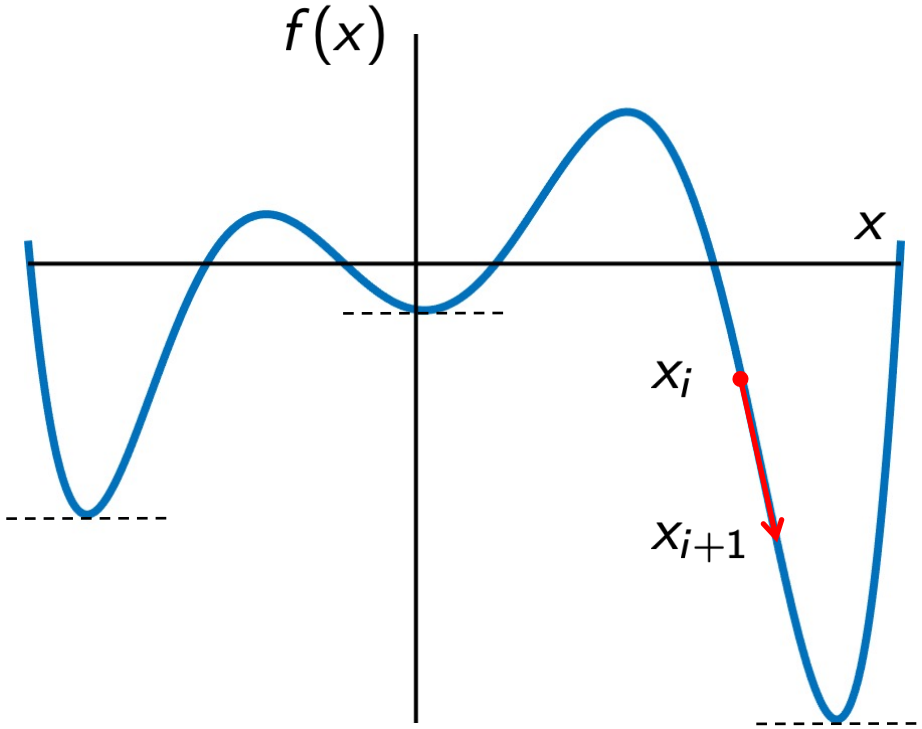
$y = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$



$y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

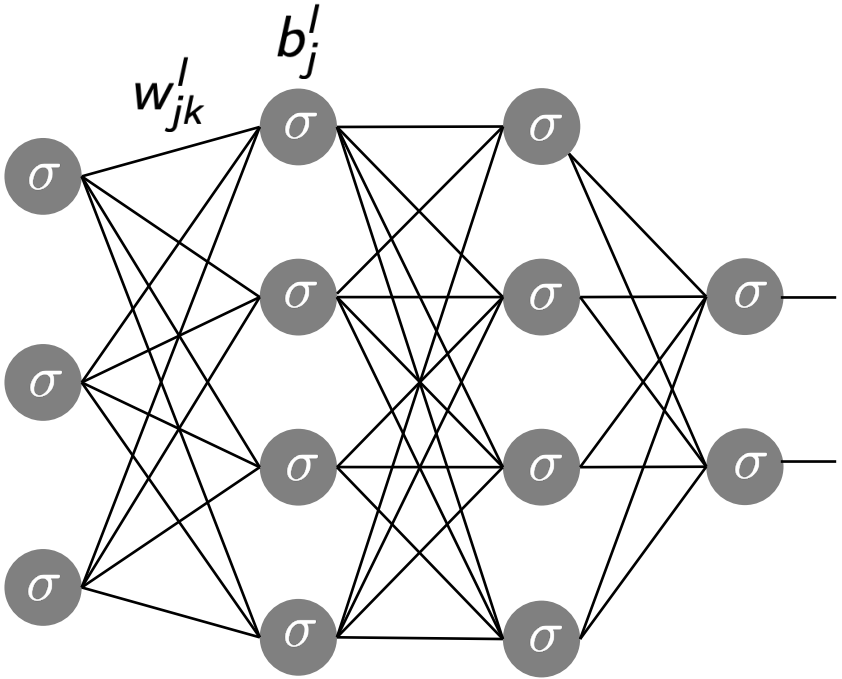
$$C(w, b) = \frac{1}{2N} \sum_{x=1}^{N} ||y_x - a_x||_2^2$$

Find minimum of function: Gradient descent



$$x_{i+1} = x_i - \alpha \frac{\partial f(x_i)}{\partial x_i}$$

Neural network training: Gradient descent



$$\min_{w,b} C(w,b)$$

$$\tilde{w}_{jk}^{l} = w_{jk}^{l} - \alpha \frac{\partial C}{\partial w_{jk}^{l}}$$

$$\tilde{b}_{j}^{l} = b_{j}^{l} - \alpha \frac{\partial C}{\partial b_{j}^{l}}$$

→ Chain rule

Backpropagation

Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

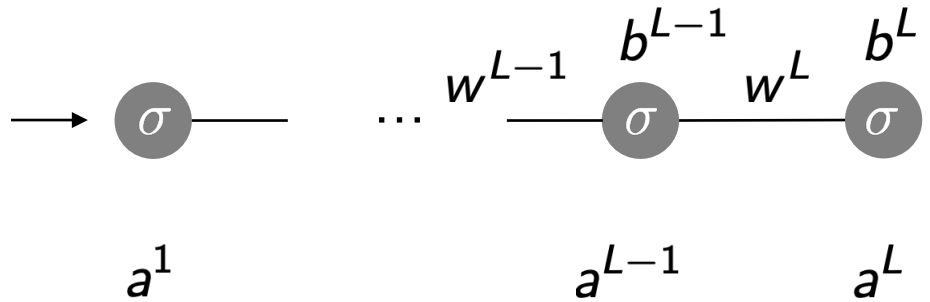
$$\frac{\partial C}{\partial w^l_{jk}} \qquad \frac{\partial C}{\partial b^l_j}$$

→ Efficient recursion to calculate these partial derivatives

Rumelhart (1986)

Backpropagation: Forward pass

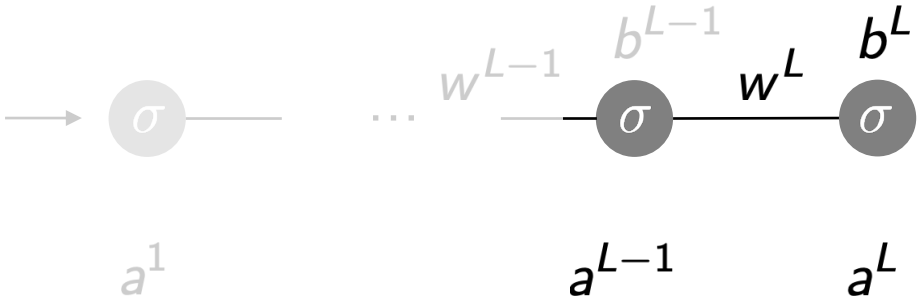
$$y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$



$$C_0 = \frac{1}{2}(y - a^L)^2$$

Backpropagation: Backward pass

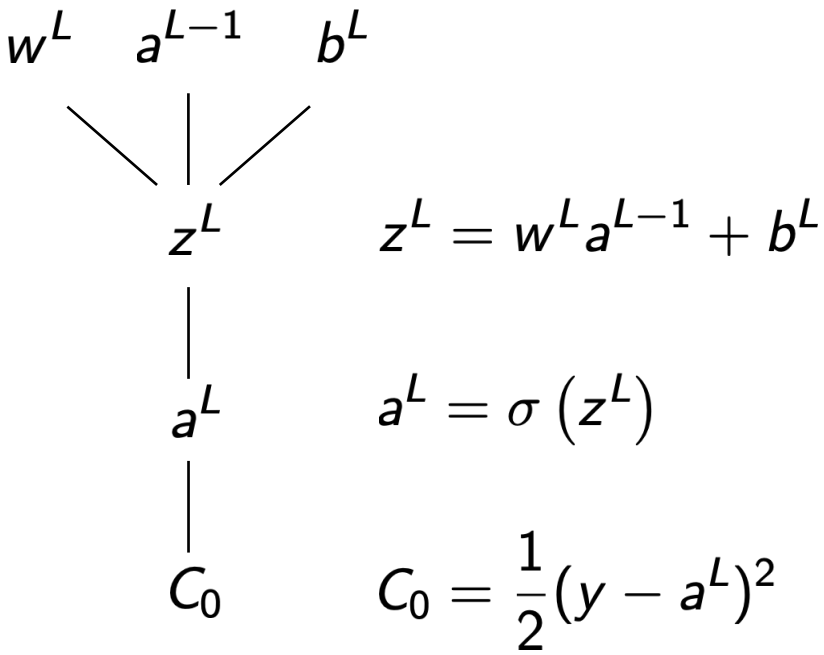
$$a^L = \sigma\underbrace{\left(w^L a^{L-1} + b^L\right)}_{z^L}$$



$$C_0 = \frac{1}{2}(y - a^L)^2$$

$$y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

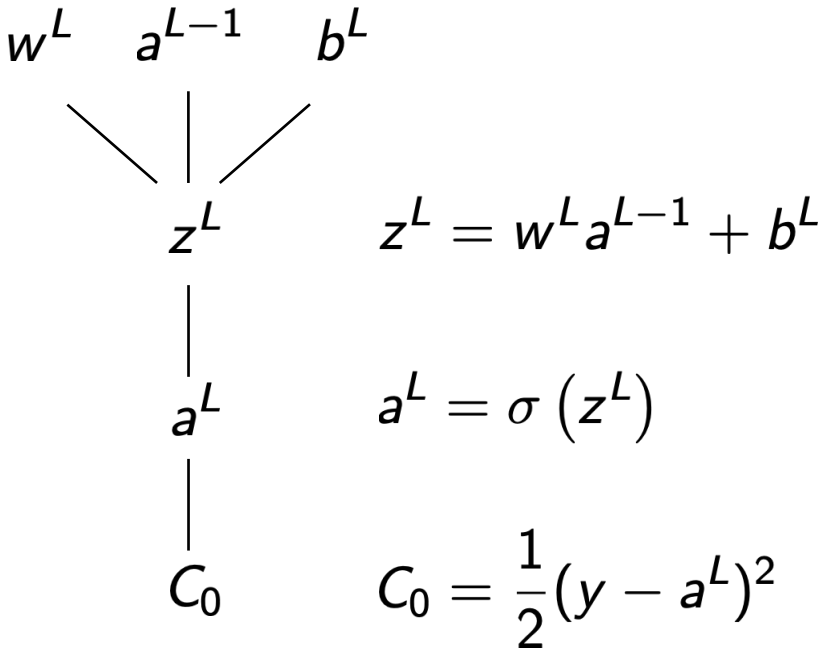
Backpropagation: Backward pass



Partial derivatives

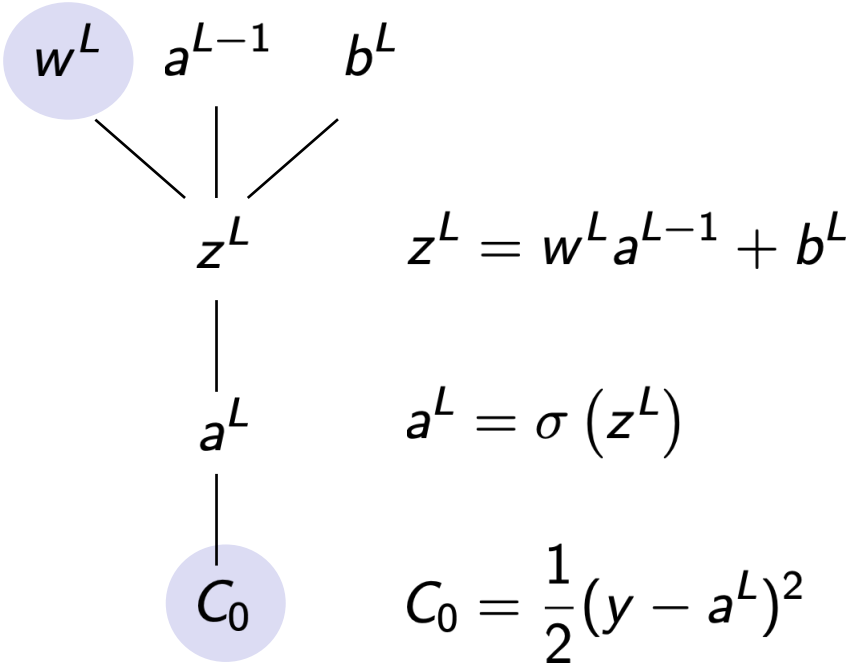
$$\frac{\partial C_0}{\partial w^L} = ?$$

$$\frac{\partial C_0}{\partial b^L} = ?$$



Backpropagation: Weights

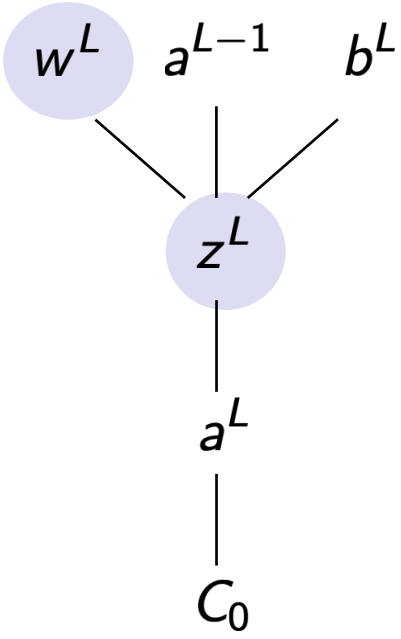
$$\frac{\partial C_0}{\partial w^L}$$



Backpropagation: Weights

$$\frac{\partial C_0}{\partial w^L} = \frac{\partial z^L}{\partial w^L}$$

$$\frac{\partial z^L}{\partial w^L} = a^{L-1}$$



$$z^L = w^L a^{L-1} + b^L$$

$$a^L = \sigma\left(z^L\right)$$

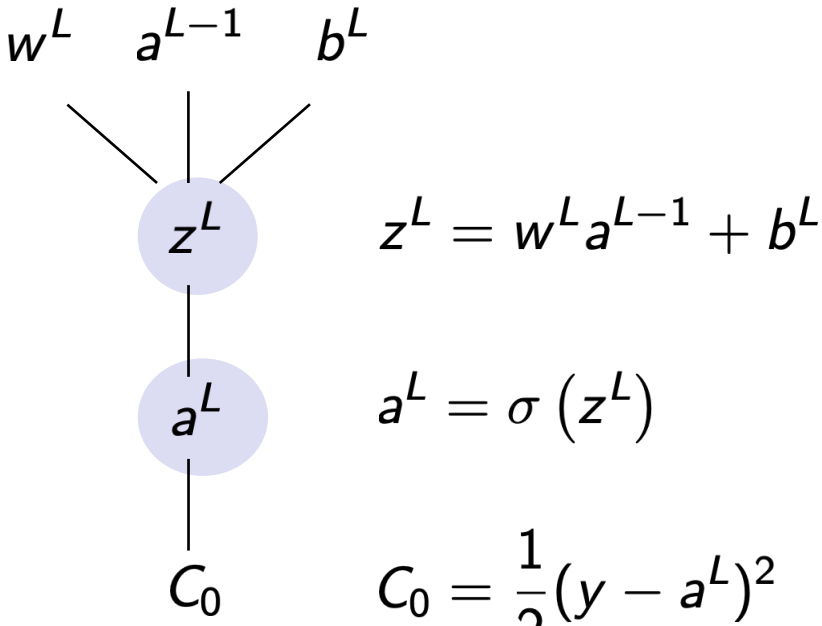
$$C_0 = \frac{1}{2}(y - a^L)^2$$

Backpropagation: Weights

$$\frac{\partial C_0}{\partial w^L} = \frac{\partial z^L}{\partial w^L}\frac{\partial a^L}{\partial z^L}$$

$$\frac{\partial z^L}{\partial w^L} = a^{L-1}$$

$$\frac{\partial a^L}{\partial z^L} = \sigma'(z^L)$$



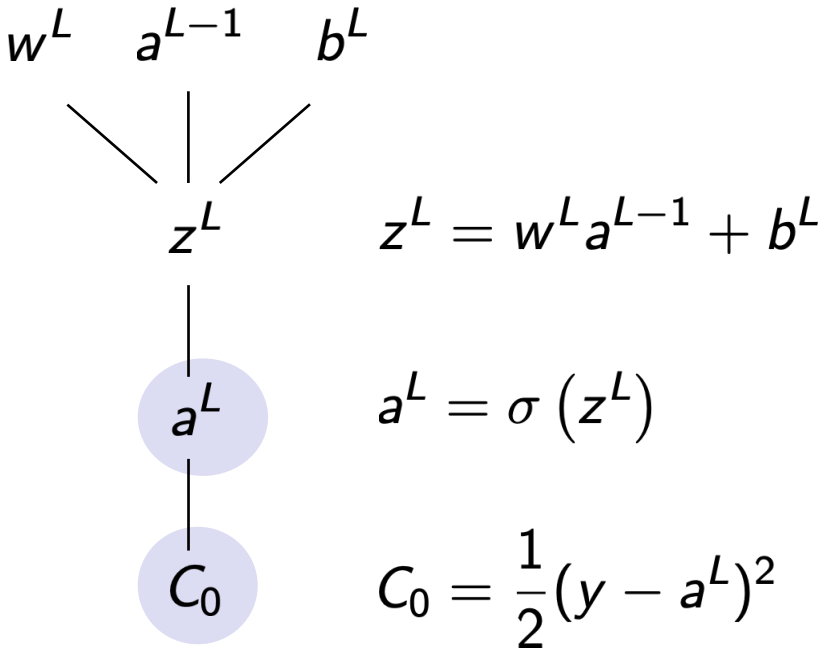
Backpropagation: Weights

$$\frac{\partial C_0}{\partial w^L} = \frac{\partial z^L}{\partial w^L}\frac{\partial a^L}{\partial z^L}\frac{\partial C_0}{\partial a^L}$$

$$\frac{\partial z^L}{\partial w^L} = a^{L-1}$$

$$\frac{\partial a^L}{\partial z^L} = \sigma'(z^L)$$

$$\frac{\partial C_0}{\partial a^L} = a^L - y$$



$$z^L = w^L a^{L-1} + b^L$$

$$a^L = \sigma\left(z^L\right)$$

$$C_0 = \frac{1}{2}(y - a^L)^2$$

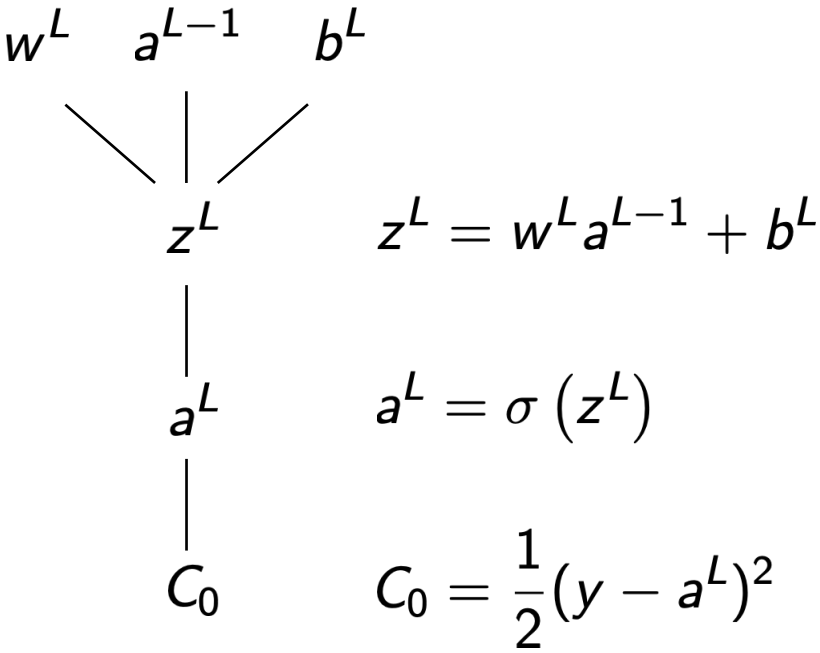
Backpropagation: Weights

$$\frac{\partial C_0}{\partial w^L} = \frac{\partial z^L}{\partial w^L}\frac{\partial a^L}{\partial z^L}\frac{\partial C_0}{\partial a^L} = a^{L-1}\sigma'(z^L)(a^L - y)$$

$$\frac{\partial z^L}{\partial w^L} = a^{L-1}$$

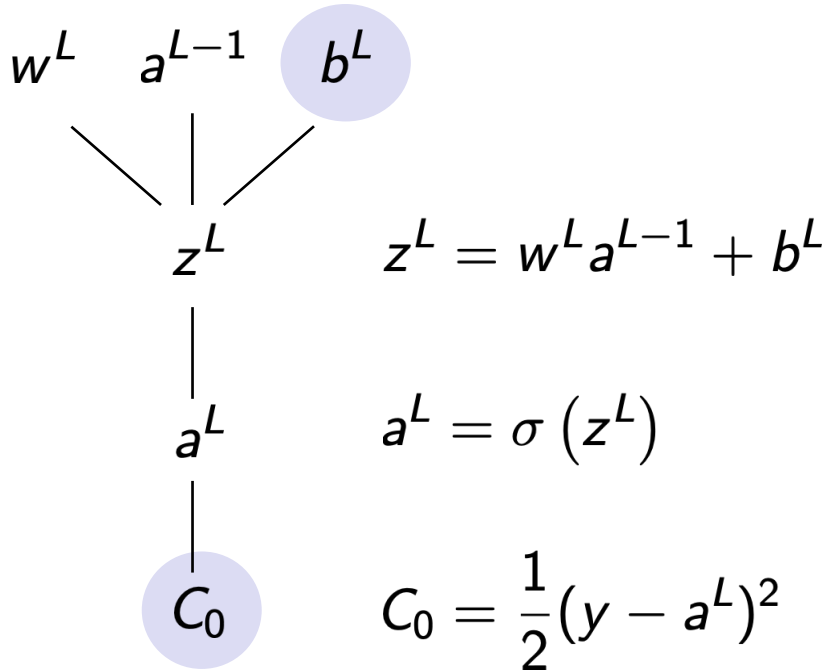
$$\frac{\partial a^L}{\partial z^L} = \sigma'(z^L)$$

$$\frac{\partial C_0}{\partial a^L} = a^L - y$$



Backpropagation: Bias

$$\frac{\partial C_0}{\partial b^L} = \frac{\partial a^L}{\partial z^L}\frac{\partial C_0}{\partial a^L} = \sigma'(z^L)(a^L - y)$$

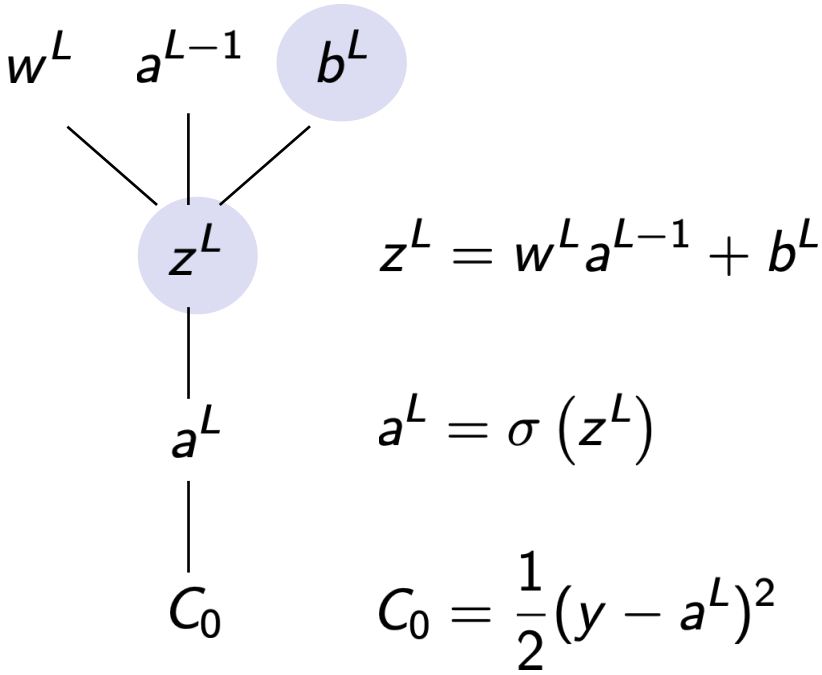


$$\frac{\partial a^L}{\partial z^L} = \sigma'(z^L)$$

$$\frac{\partial C_0}{\partial a^L} = a^L - y$$

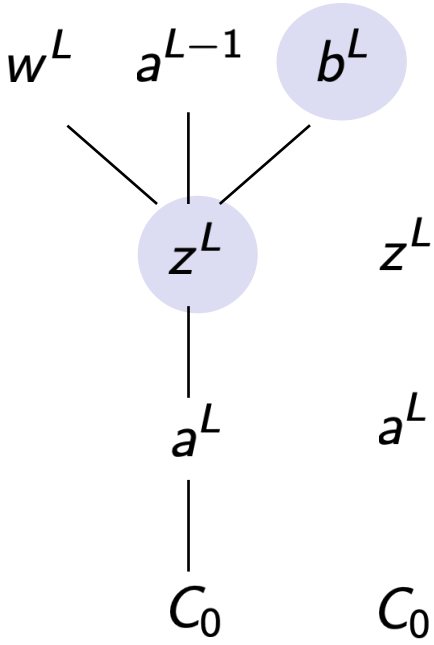
Backpropagation: Bias

$$\frac{\partial C_0}{\partial b^L} = \frac{\partial z^L}{\partial b^L}\frac{\partial a^L}{\partial z^L}\frac{\partial C_0}{\partial a^L} = \sigma'(z^L)(a^L - y)$$



Backpropagation: Bias

$$\frac{\partial C_0}{\partial b^L} = \frac{\partial z^L}{\partial b^L}\frac{\partial a^L}{\partial z^L}\frac{\partial C_0}{\partial a^L} = 1\sigma'(z^L)(a^L - y)$$



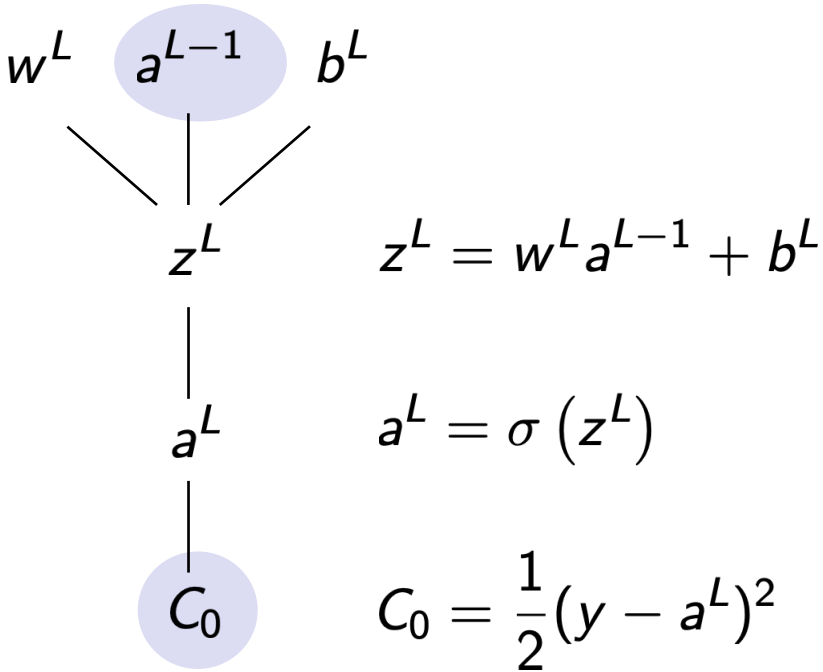
$$z^L = w^L a^{L-1} + b^L$$

$$a^L = \sigma\left(z^L\right)$$

$$C_0 = \frac{1}{2}(y - a^L)^2$$

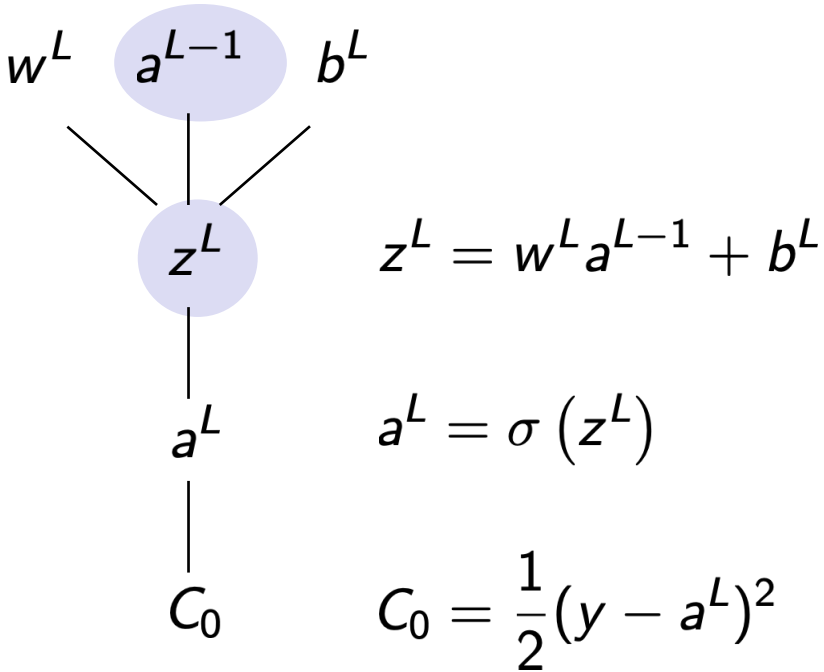
Backpropagation: Activation of previous layer

$$\frac{\partial C_0}{\partial a^{L-1}} = \frac{\partial a^L}{\partial z^L}\frac{\partial C_0}{\partial a^L} = \sigma'(z^L)(a^L - y)$$



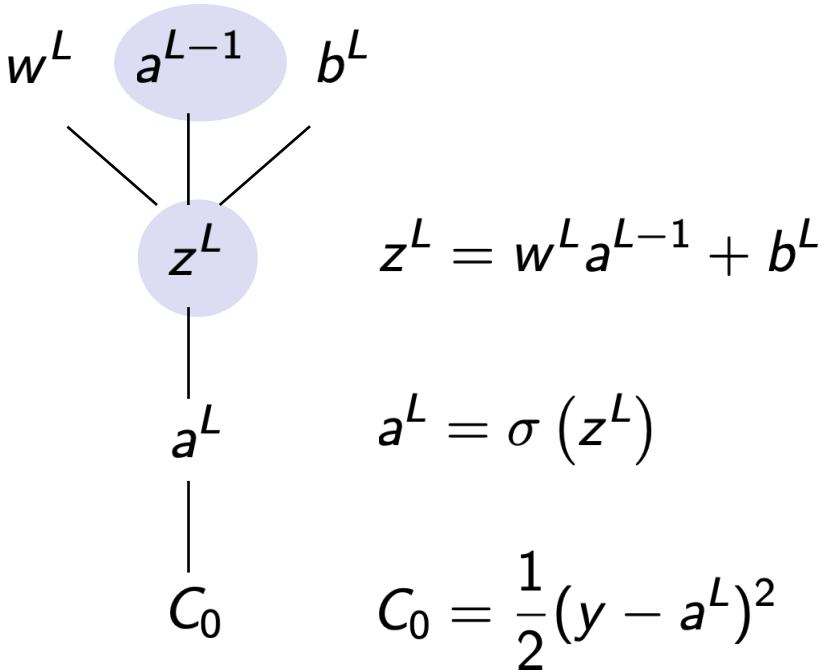
Backpropagation: Activation of previous layer

$$\frac{\partial C_0}{\partial a^{L-1}} = \frac{\partial z^L}{\partial a^{L-1}} \frac{\partial a^L}{\partial z^L} \frac{\partial C_0}{\partial a^L} = \sigma'(z^L)(a^L - y)$$



Backpropagation: Activation of previous layer

$$\frac{\partial C_0}{\partial a^{L-1}} = \frac{\partial z^L}{\partial a^{L-1}} \frac{\partial a^L}{\partial z^L} \frac{\partial C_0}{\partial a^L} = w^L \sigma'(z^L)(a^L - y)$$

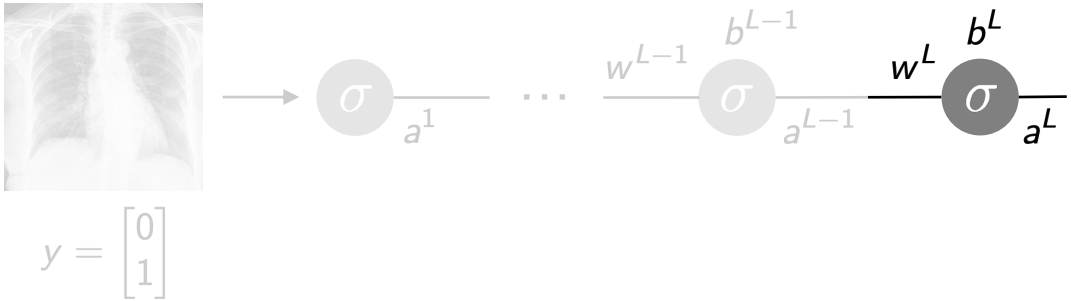


Backpropagation

$$\frac{\partial C_0}{\partial w^L} = a^{L-1}\sigma'(z^L)\frac{\partial C_0}{\partial a^L}$$

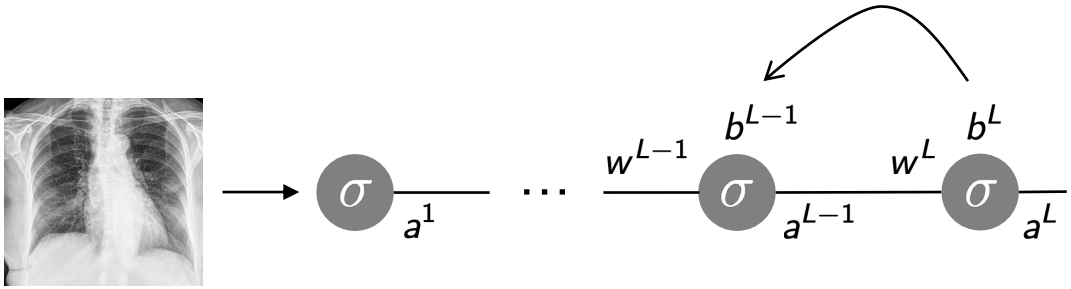
$$\frac{\partial C_0}{\partial b^L} = \sigma'(z^L)\frac{\partial C_0}{\partial a^L}$$

$$\frac{\partial C_0}{\partial a^L} = a^L - y$$



Backpropagation

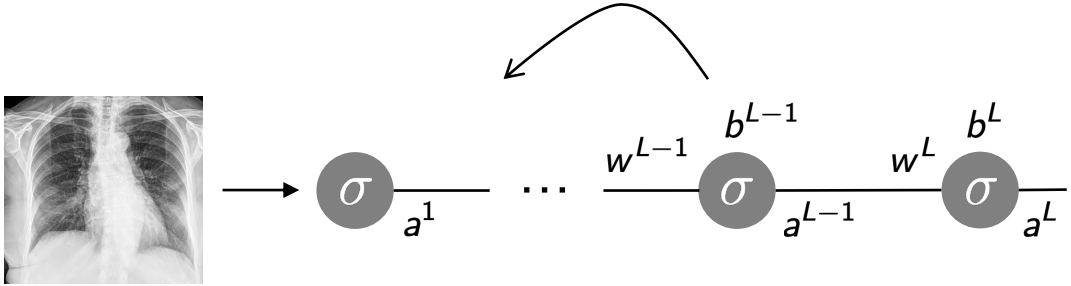
$$\frac{\partial C_0}{\partial w^{L-1}} = a^{L-2}\sigma'(z^{L-1})\frac{\partial C_0}{\partial a^{L-1}} \qquad \frac{\partial C_0}{\partial b^{L-1}} = \sigma'(z^{L-1})\frac{\partial C_0}{\partial a^{L-1}} \qquad \frac{\partial C_0}{\partial a^{L-1}} = w^{L}\sigma'(z^{L})\frac{\partial C_0}{\partial a^{L}}$$



$$y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Backpropagation

$$\frac{\partial C_0}{\partial w^l} = a^{l-1}\sigma'(z^l)\frac{\partial C_0}{\partial a^l} \qquad \frac{\partial C_0}{\partial b^l} = \sigma'(z^l)\frac{\partial C_0}{\partial a^l} \qquad \frac{\partial C_0}{\partial a^l} = w^{l+1}\sigma'(z^{l+1})\frac{\partial C_0}{\partial a^{l+1}}$$



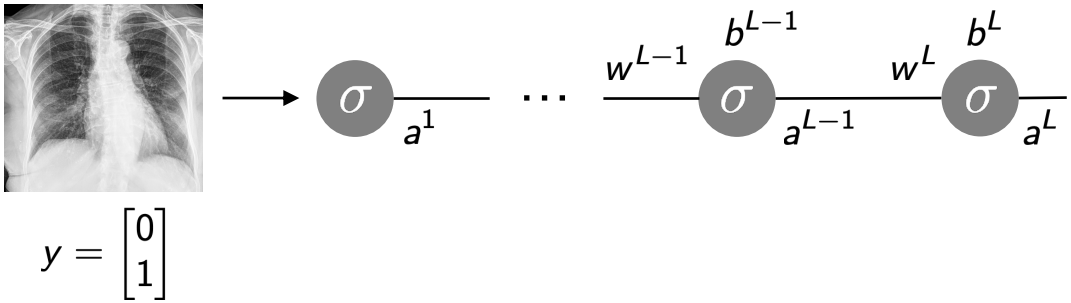
$$y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Plug partial derivatives into gradient descent

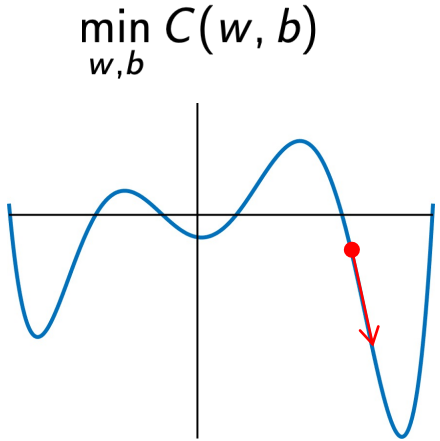
$$\frac{\partial C_0}{\partial w^l} = a^{l-1}\sigma'(z^l)\frac{\partial C_0}{\partial a^l}$$

$$\frac{\partial C_0}{\partial b^l} = \sigma'(z^l)\frac{\partial C_0}{\partial a^l}$$

$$\frac{\partial C_0}{\partial a^l} = w^{l+1}\sigma'(z^{l+1})\frac{\partial C_0}{\partial a^{l+1}}$$



$$y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$



Gradient descent

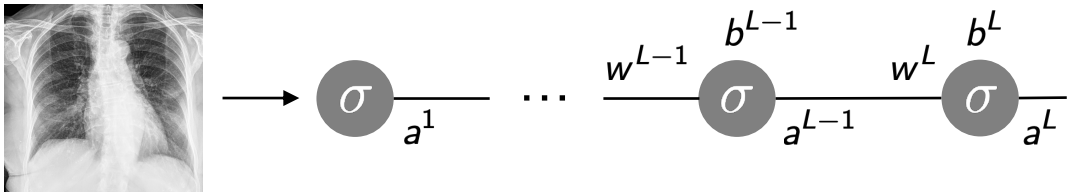
Loop over training examples

$$\frac{\partial C_0}{\partial w^l} = a^{l-1}\sigma'(z^l)\frac{\partial C_0}{\partial a^l}$$

$$\frac{\partial C_0}{\partial b^l} = \sigma'(z^l)\frac{\partial C_0}{\partial a^l}$$

$$\frac{\partial C_0}{\partial a^l} = w^{l+1}\sigma'(z^{l+1})\frac{\partial C_0}{\partial a^{l+1}}$$

$$\min_{w,b} C(w,b)$$



$$y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$y = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

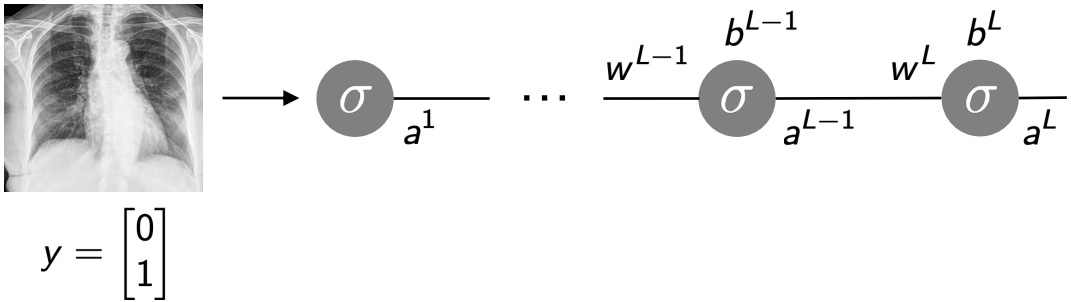
$\cdots$

$$y = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Backpropagation: Efficiency and insights

$$\frac{\partial C_0}{\partial w^l} = a^{l-1}\sigma'(z^l)\frac{\partial C_0}{\partial a^l} \qquad \frac{\partial C_0}{\partial b^l} = \sigma'(z^l)\frac{\partial C_0}{\partial a^l} \qquad \frac{\partial C_0}{\partial a^l} = w^{l+1}\sigma'(z^{l+1})\frac{\partial C_0}{\partial a^{l+1}}$$



$$y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

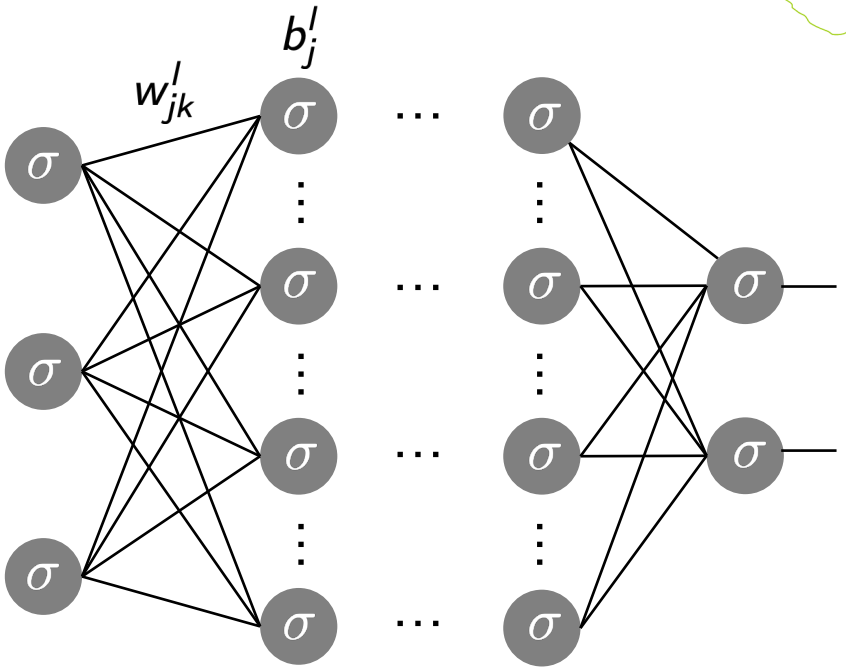
- Each computation involves just two layers
- Avoid recomputing identical expressions in the chain rule
- Gradients provide insight into what determines speed of learning

Backpropagation: General formulation

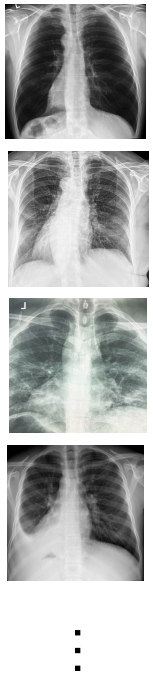
$$\frac{\partial C_0}{\partial w^l_{jk}} = a^{l-1}_k \sigma'(z^l_j) \frac{\partial C_0}{\partial a^l_j}$$

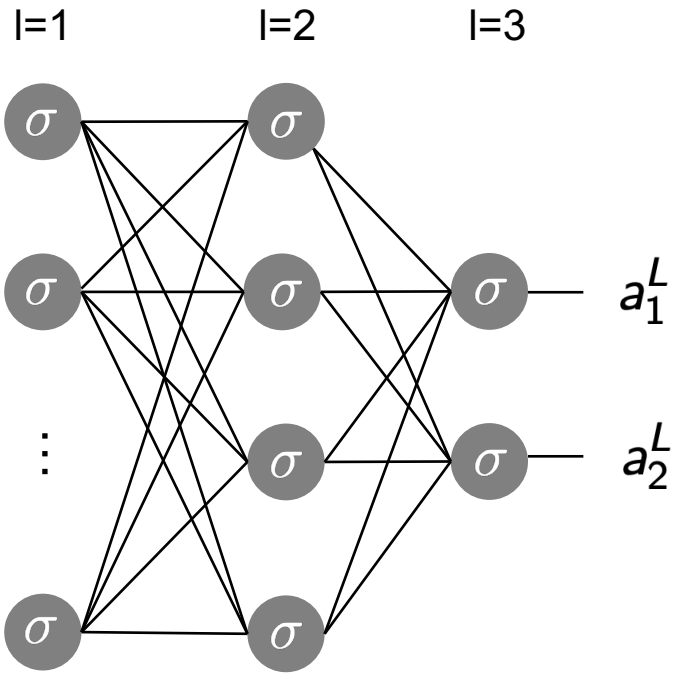
$$\frac{\partial C_0}{\partial b^l_j} = \sigma'(z^l_j) \frac{\partial C_0}{\partial a^l_j}$$

$$\frac{\partial C_0}{\partial a^l_j} = \sum_j w^{l+1}_{jk} \sigma'(z^{l+1}_j) \frac{\partial C_0}{\partial a^{l+1}_j}$$



Back to our chest X-ray problem

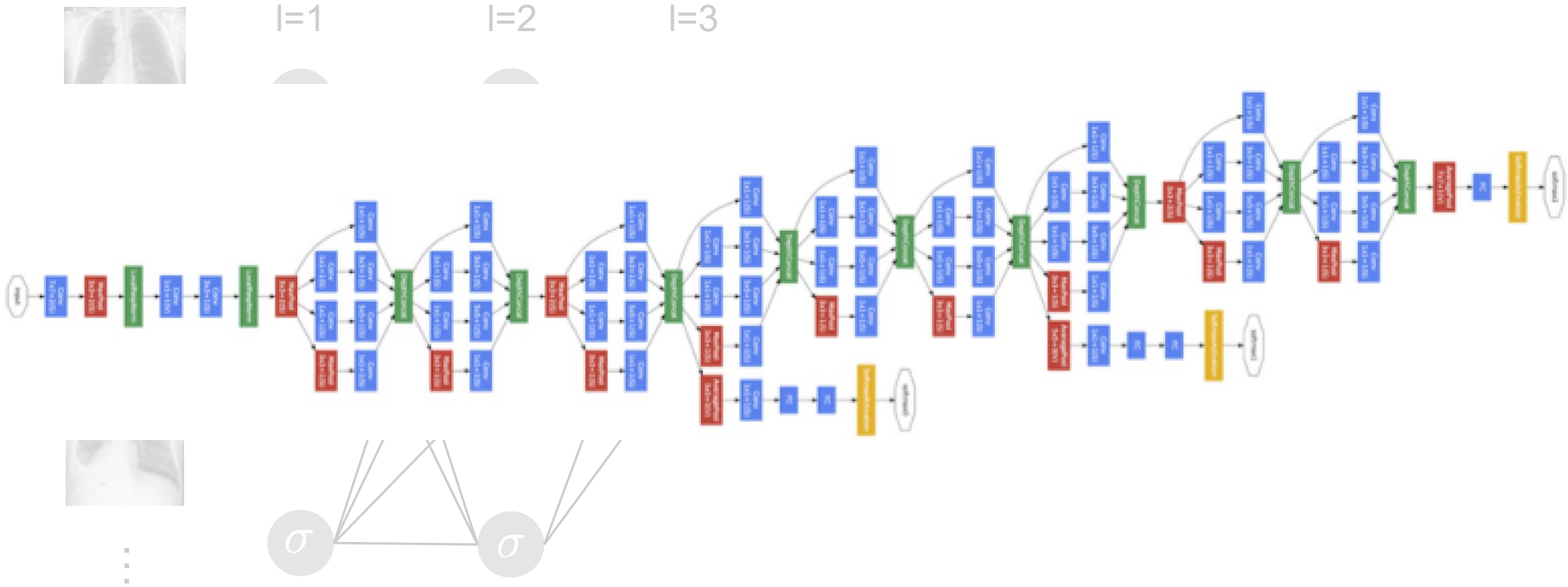




$$\min_{w,b} C(w, b)$$

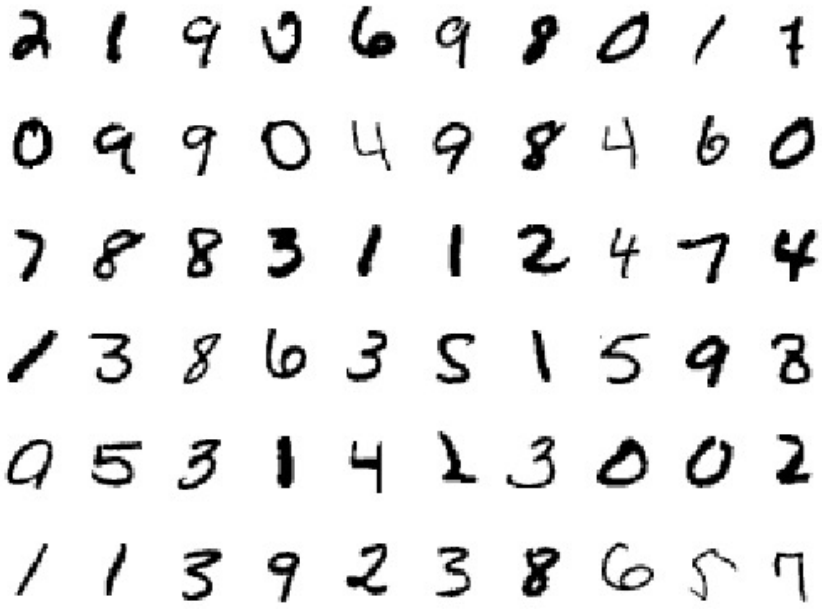
Accuracy

Back to our chest X-ray problem



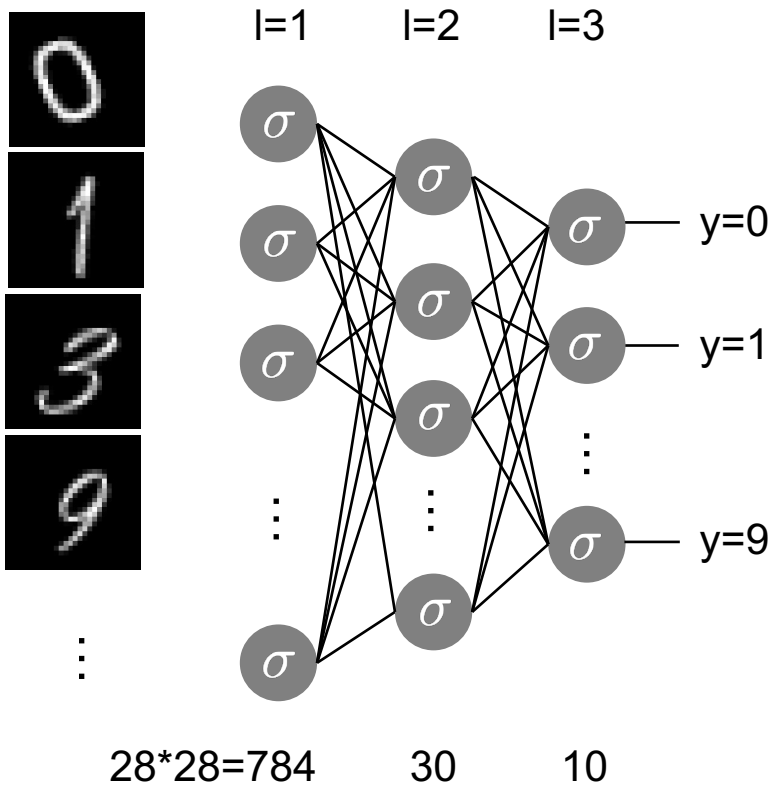
Szegedy CVPR 2015

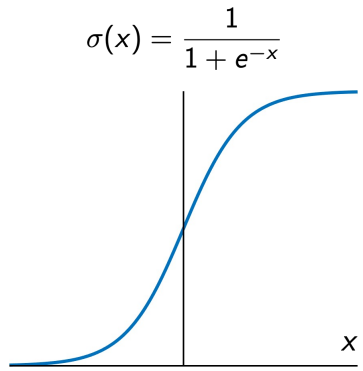
Handwritten character recognition: MNIST



LeCun (1999)

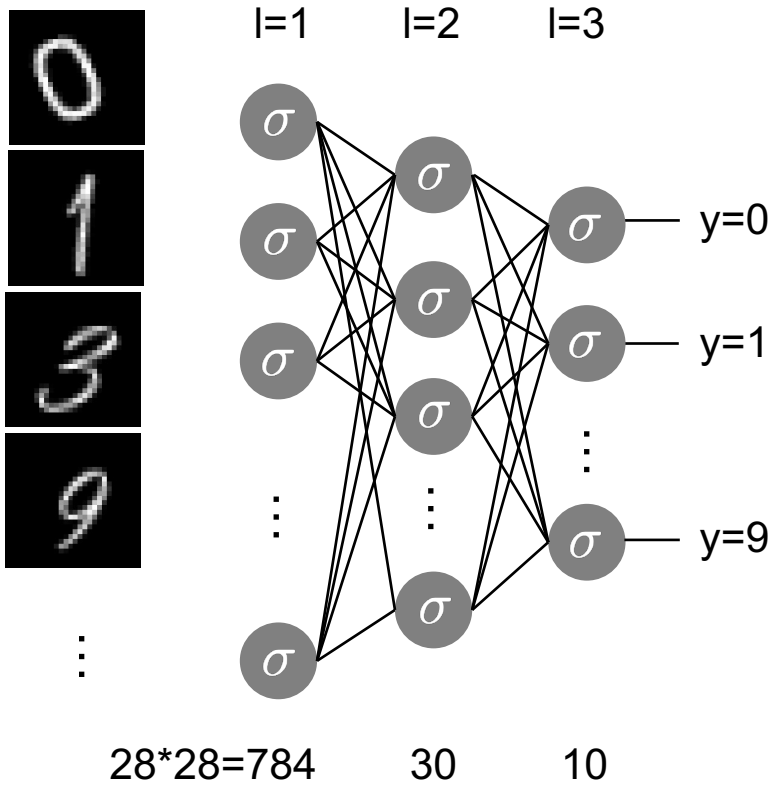
Adding more layers to our network?



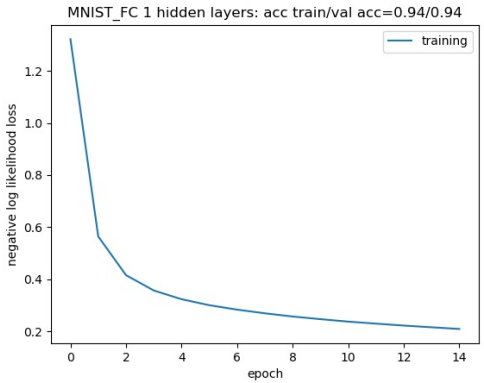


Sigmoid nonlinearity

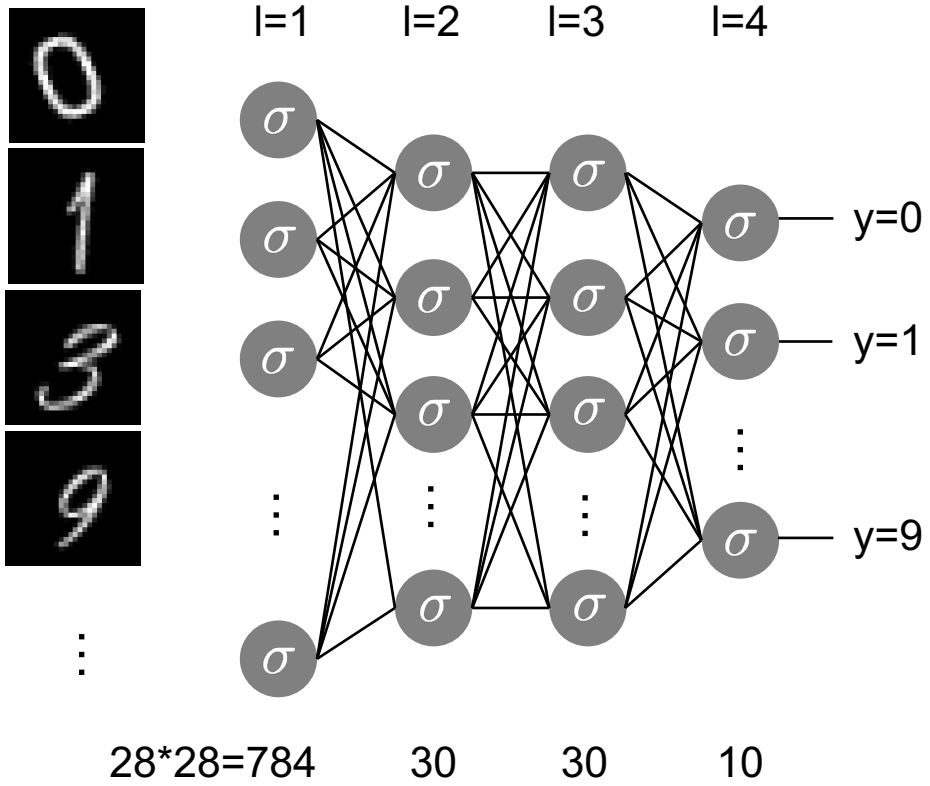
Adding more layers to our network?



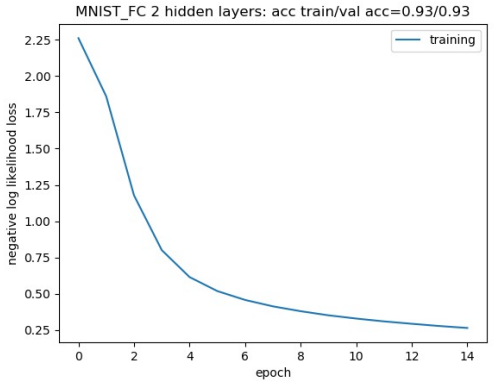
$$\min_{w,b} C(w, b)$$



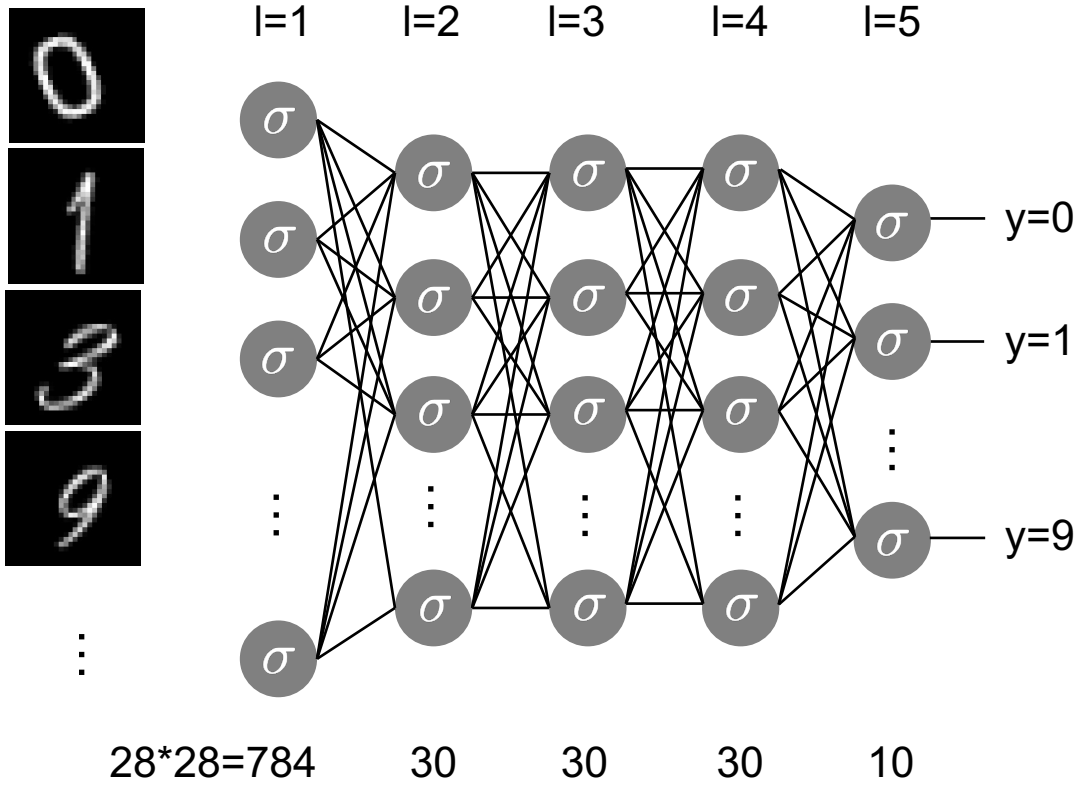
Adding more layers to our network?



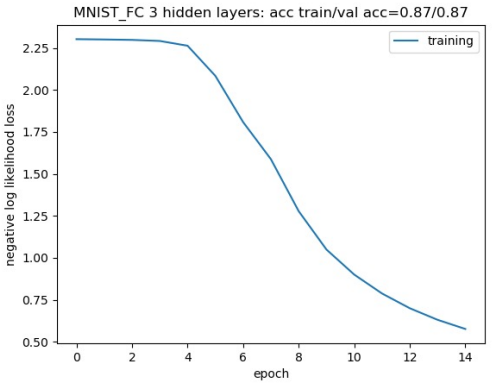
$$\min_{w,b} C(w, b)$$



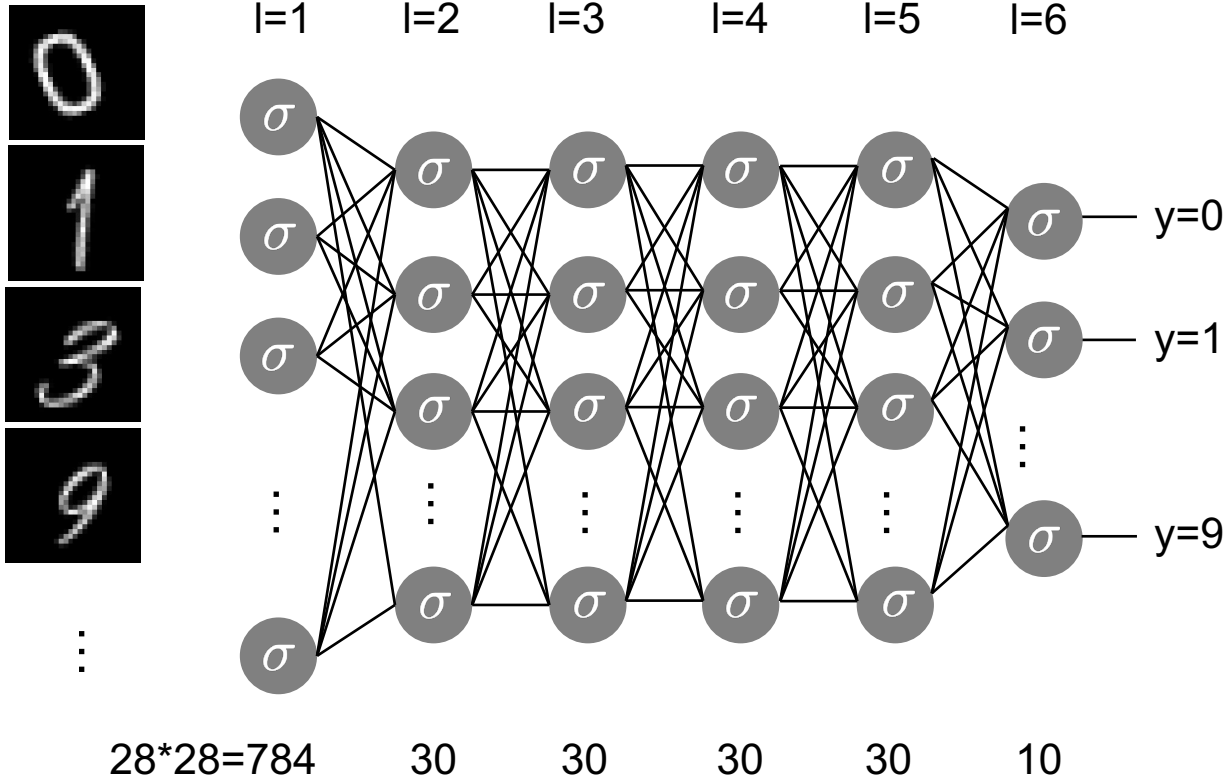
Adding more layers to our network?



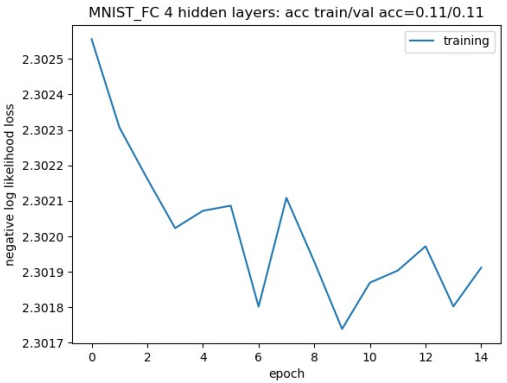
$$\min_{w,b} C(w, b)$$



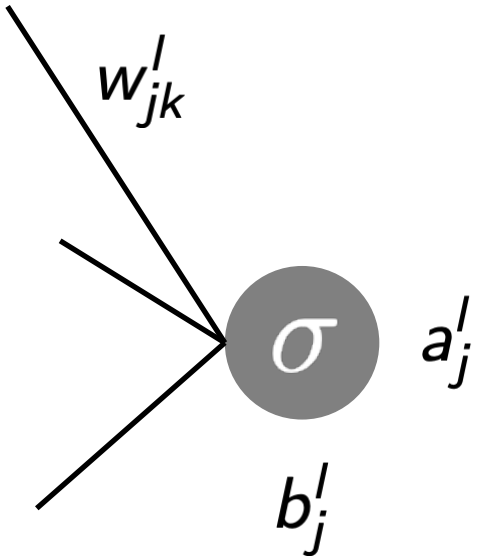
Adding more layers to our network?



$$\min_{w,b} C(w, b)$$



Insights from backpropagation gradients

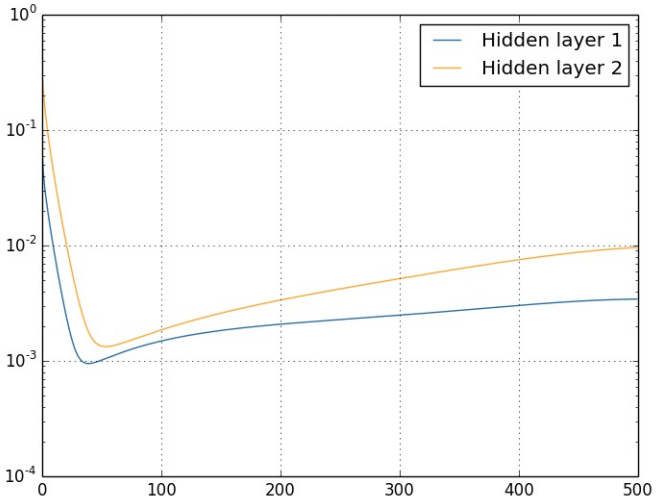


$$\left\| \frac{\partial C_0}{\partial b^l} \right\|_2 = \left\| \sigma'(z^l) \frac{\partial C_0}{\partial a^l} \right\|_2$$

Strength of gradients determines how fast neurons learn

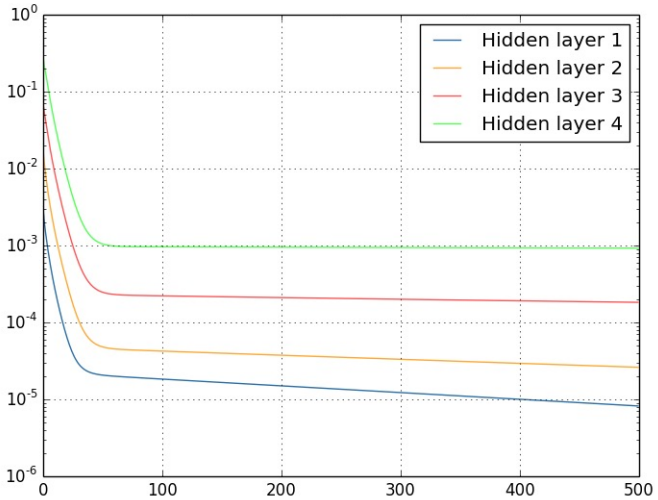
$$\left\| \frac{\partial C_0}{\partial b^l} \right\|_2$$

2 hidden layers



Training iterations

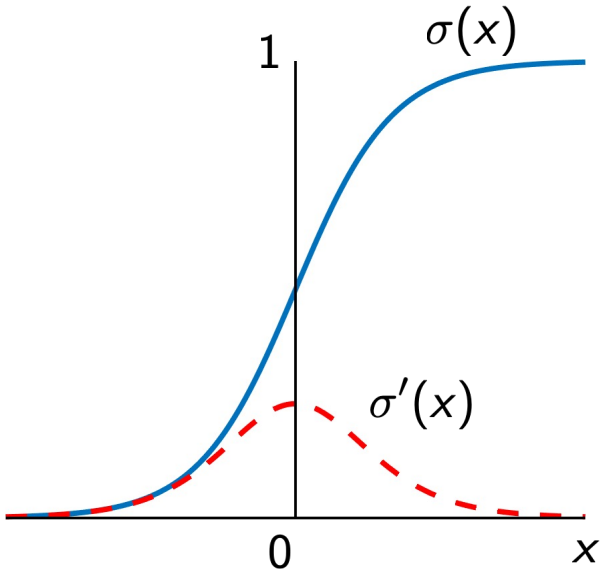
4 hidden layers



Training iterations

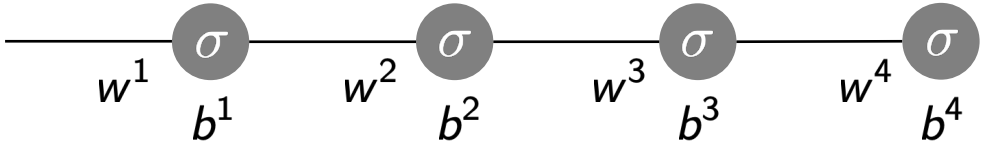
Nielsen (2015)

Vanishing gradient problem



$$\frac{\partial C_0}{\partial b_j^l} = \sigma'(z_j^l)\frac{\partial C_0}{\partial a_j^l}$$

$$\frac{\partial C_0}{\partial b^1} = \sigma'(z^1)\cdot w^2\cdot\sigma'(z^2)\cdot w^3\cdot\sigma'(z^3)\cdot w^4\cdot\sigma'(z^3)\cdot\frac{\partial C_0}{\partial a^4}$$

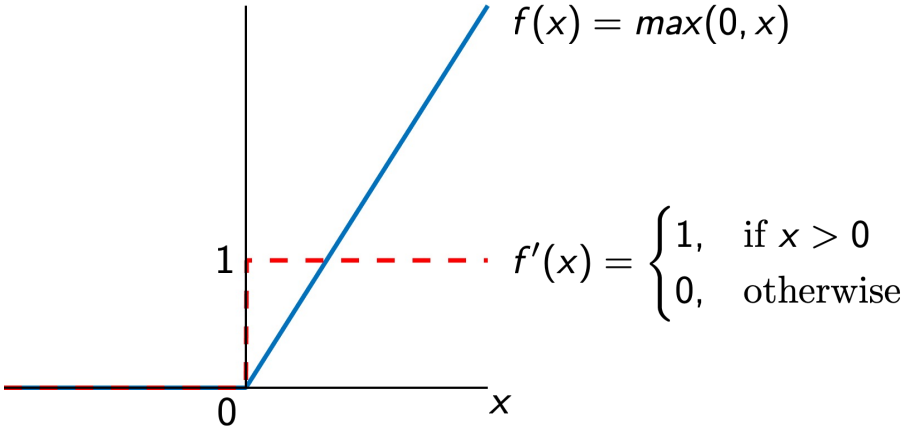


max(sigmoid gradient)=1/4,太小了，
层数越多，累乘起来越接近0，导致梯度消失

Neurons get adjusted at very different rates ⟶ Gradient instability

Ongoing research

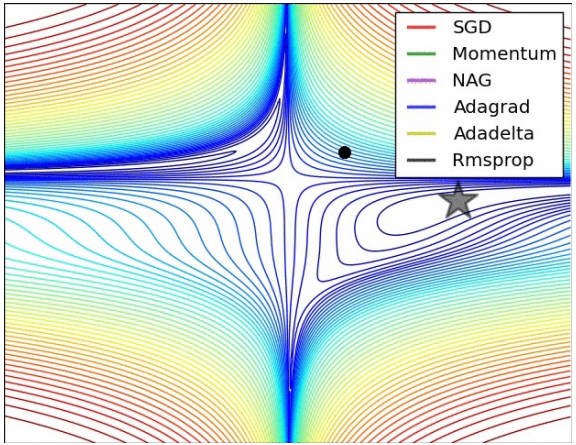
- Nonlinearities: ReLU and variants,…



Hahnloser 2000

Ongoing research

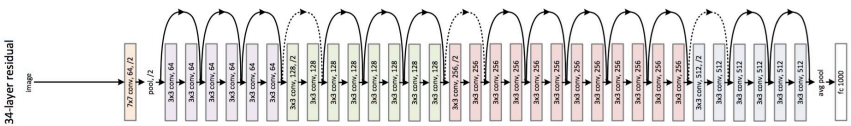
- Nonlinearities
- Optimization algorithms (ADAM, Adagrad, RMSProp,…)

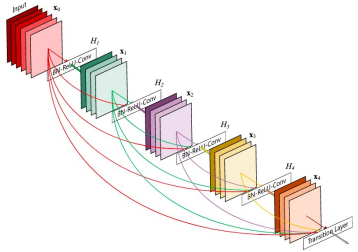


Kingma 2014
Duchi 2011
Hinton 2013
Ruder 2016

Ongoing research

- Nonlinearities
- Optimization algorithms (ADAM, Adagrad, RMSProp,…)
- Network architectures





He CVPR 2016
Huang 2016

Summary

- Short recap of neural networks
- Training neural networks with gradient descent
- Backpropagation: Efficient implementation of chain rule
- Deep neural networks: Gradient instability

Exercise

Computational MR imaging
Laboratory 10: Training neural networks with backpropagation

Report is due on Wednesday the week after the lab session at 23:59. Send your report by email to Bruno Riemenschneider (bruno.riemenschneider@fau.de) and Florian Knoll (florian.knoll@fau.de).

Learning objectives

- Train a fully connected network for MNIST classification in Pytorch
- Change the number of hidden layers from 1 to 5
- Observe the
- Examine effects of dropping data on neural network performance
- Learn about effects of over and underfitting

1. **Training a neural network for MNIST classification**

Open the script MNIST_classification_pytorch_1p5_cpu.py. This script serves as a template that handles file I/O and plotting.

   i. Familiarize yourself with the script, the data, and the labels. Plot an example of selected training images.
   ii. Define a fully connected neural network architecture with one hidden layer with 30 neurons and train it. Use the sigmoid activation function for input and the hidden layers, and log softmax for the output layer. Recommended training settings are:
      i. Batch size: 64
      ii. Loss function: NLLLoss
      iii. Optimizer: SGD
      iv. Training epochs: 15
      v. Learning rate: 0.003
      vi. Momentum: 0.9
   iii. Plot the training loss over the epochs

2. **Investigate the impact of the number of hidden layers and the activation function**
   i. Increase the number of hidden layers to [1,5].
   ii. Change the activation function from sigmoid to ReLU for the input and hidden layers.
   iii. Plot the training loss over the epochs for all experiments and discuss your findings.