GHENT UNIVERSITY

FACULTY OF ENGINEERING AND ARCHITECTURE

E016350 - Artificial Intelligence

Lecture 5

Machine learning

Neural networks

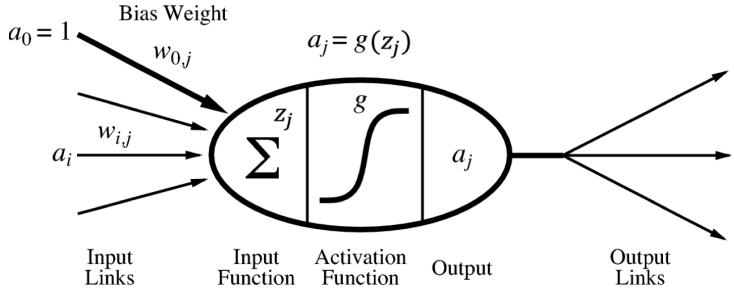
Part 1

Aleksandra Pizurica

Ghent University

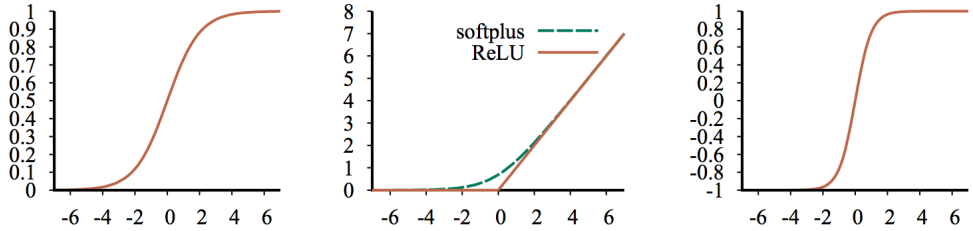
Fall 2024

McCulloch-Pitts neuron



$$a_j = g(z_j) = g\Big(\sum_i w_{i,j} a_i\Big)$$

Common activation functions in deep learning

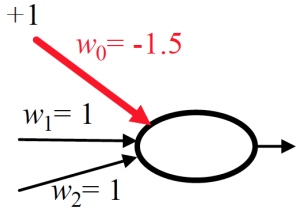


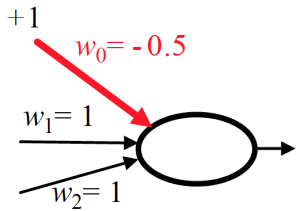
Left: *logistic* (*sigmoid*); Middle: *ReLU* and *softplus*; Right: *tanh*

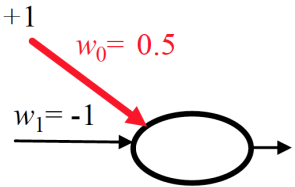
$$\sigma(x) = \frac{1}{1+e^{-x}};\ ReLU(x) = \max(0,x);\ softplus(x) = \log(1+e^{x});\ tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$$

Changing the bias weight moves the threshold location

Implementing logical gates: examples







AND

a_1	a_2	*out*
0	0	0
0	1	0
1	0	0
1	1	1

OR

a_1	a_2	*out*
0	0	0
0	1	1
1	0	1
1	1	1

NOT

in	*out*
0	1
1	0

Determining the weights

$$out = \text{sgn}(w_0 + w_1 a_1 + w_2 a_2)$$

a_1	a_2	out
0	0	0
0	1	0
1	0	0
1	1	1

$\Rightarrow$

$$\begin{aligned} w_0 + w_1 0 + w_2 0 &< 0 \\ w_0 + w_1 0 + w_2 1 &< 0 \\ w_0 + w_1 1 + w_2 0 &< 0 \\ w_0 + w_1 1 + w_2 1 &\geq 0 \end{aligned}$$

$\Rightarrow$

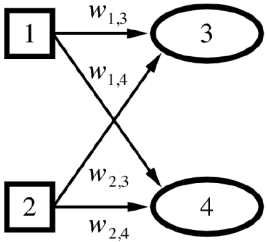
$$\begin{aligned} w_0 &< 0 \\ w_0 &< -w_2 \\ w_0 &< -w_1 \\ w_0 &\geq -w_1 - w_2 \end{aligned}$$

The solution is not unique; Moreover, there are infinitely many solutions. The same holds for OR and NOT networks.

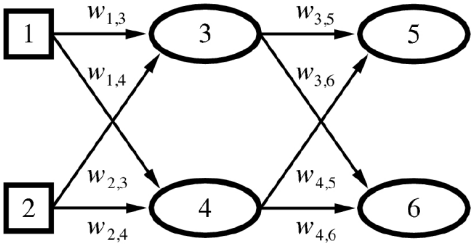
Network structures

- Feed-forward networks
  - Each node computes a function of its inputs and passes the result to its successors.
  - Information flows from the input to the output nodes; there are no loops.
  - Perceptron
    - ⋆ single-layer perceptron
    - ⋆ multi-layer perceptron
- Recurrent networks
  - The output from some nodes is fed back to the input
    - → Bi-directional flow of information
  - Can be interpreted as a dynamical system with internal state or memory

Feed-forward networks



Single layer feed forward net.

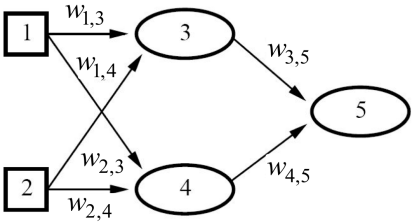


Multi layer feed forward network

Notation

- We denote by $a_j^{(l)}$ the activation of the j-th node in the l-th layer of the network, which is also the signal at the output of that node.
- The input to that same node is a weighted sum of the activations of the nodes from the previous layer: $z_j^{(l)} = \sum_i w_{i,j} a_i^{(l-1)}$
- The nonlinear activation function $g^{(l)}$ can be different in each layer l, but unless necessary to express this explicitly, we will write only g
- For compactness, we always omit the index of the layer when no confusion possible
- Thus we write $a_j = g(z_j) = g\left(\sum_i w_{i,j} a_i\right)$
- By convention, we will denote the i-th node in the input layer by x_i
  thus in the first layer: $a_j^1 = g\left(\sum_i w_{i,j} x_i\right) = g(\mathbf{w}_j \cdot \mathbf{x})$

Feed-forward networks - example

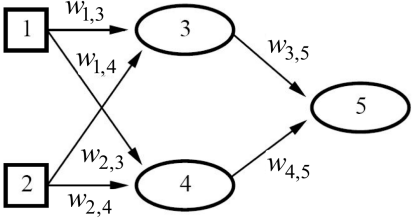


Feed-forward network = a parameterized family of nonlinear functions:

$$\begin{aligned} a_5 &= g(w_{3,5}a_3 + w_{4,5}a_4) \\ &= g\Big(w_{3,5}g(w_{1,3}x_1 + w_{2,3}x_2) + w_{4,5}g(w_{1,4}x_1 + w_{2,4}x_2)\Big) \end{aligned}$$

Adjusting weights changes the function: do learning this way!

Feed-forward networks - example - vector notation

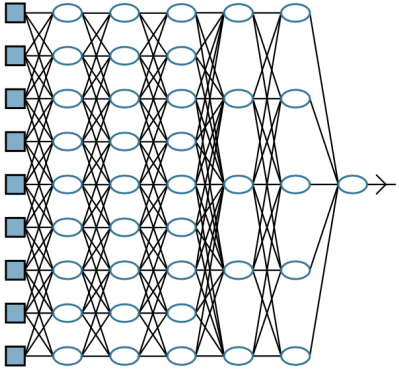


$$a_5 = g\left(\begin{bmatrix} w_{3,5} & w_{4,5} \end{bmatrix} \begin{bmatrix} a_3 \\ a_4 \end{bmatrix}\right) = g\Big(\underbrace{\begin{bmatrix} w_{3,5} & w_{4,5} \end{bmatrix}}_{\mathbf{W}^{(2)}} g\Big(\underbrace{\begin{bmatrix} w_{1,3} & w_{2,3} \\ w_{1,4} & w_{2,4} \end{bmatrix}}_{\mathbf{W}^{(1)}} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\Big)\Big)$$

$$= g\Big(\mathbf{W}^{(2)} g\Big(\mathbf{W}^{(1)}\mathbf{x}\Big)\Big) = g\Big(\mathbf{w}^{(2)} \cdot g\Big(\mathbf{W}^{(1)}\mathbf{x}\Big)\Big)$$

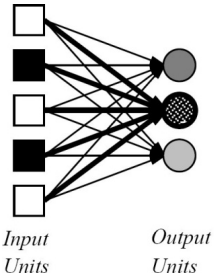
$\mathbf{W}^{(l)}$ is the **weight matrix** in the l-th layer. Its rows are the weight vectors.
Omitting the layer index: $\mathbf{W}(j,:) = \mathbf{w}_j{}^\top = [w_{0,j} \dots w_{n,j}]$.

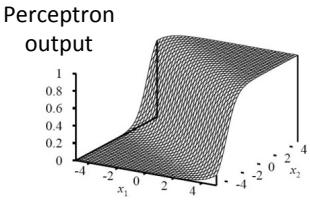
Feed-forward network with n layers



$$h_{\mathbf{w}}(\mathbf{x}) = g^{(n)}\left(\mathbf{W}^{(n)} g^{(n-1)}\left(\mathbf{W}^{(n-1)} \ \ldots g^{(1)}\left(\mathbf{W}^{(1)}\mathbf{x}\right)\right)\right)$$

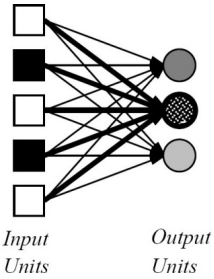
Single layer feed-forward networks

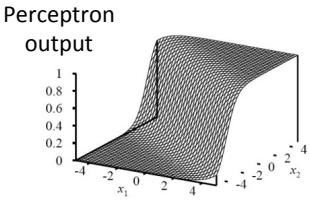




Output units all operate separately - no shared weights
Adjusting weights moves the location, orientation, and steepness of cliff

Single layer feed-forward networks





Output units all operate separately - no shared weights
Adjusting weights moves the location, orientation, and steepness of cliff

Single layer perceptrons can solve only **linearly separable** cases. **Why?**

Reminder: Linear classifier with a hard threshold

$$h_{\mathbf{w}}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Perceptron learning rule:

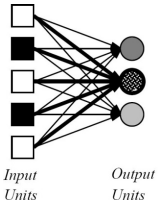
$$w_i \leftarrow w_i + \alpha(y - h_{\mathbf{w}}(\mathbf{x}))x_i$$

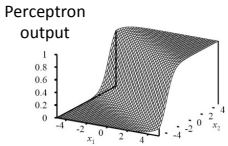
Converges to the perfect linear separator (if data are linearly separable)
Note that we have the same form of the update rule for logistic regression.

In essence, this rule says:

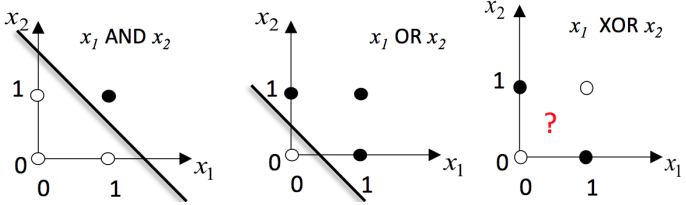
- if the output is correct: (i.e., $h_{\mathbf{w}} = y$): do nothing!
- otherwise push each w_i in the right direction:
  - If $y = 1$ but $h_{\mathbf{w}} = 0$: increase w_i if $x_i > 0$ and decrease w_i if $x_i < 0$
  - If $y = 0$ but $h_{\mathbf{w}} = 1$: decrease w_i if $x_i > 0$ and increase w_i if $x_i < 0$

Single layer feed-forward networks

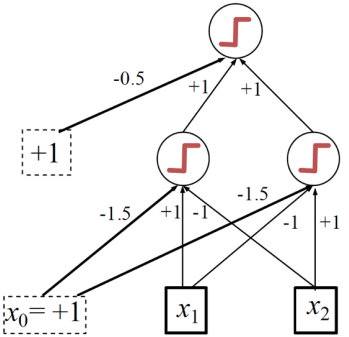




Represents a linear separator in input space $\sum_i w_i x_i > 0$ or $\mathbf{w} \cdot \mathbf{x} > 0$

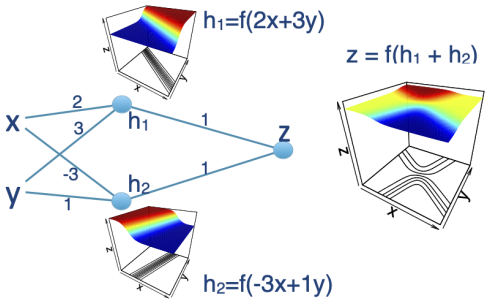


Multilayer perceptron



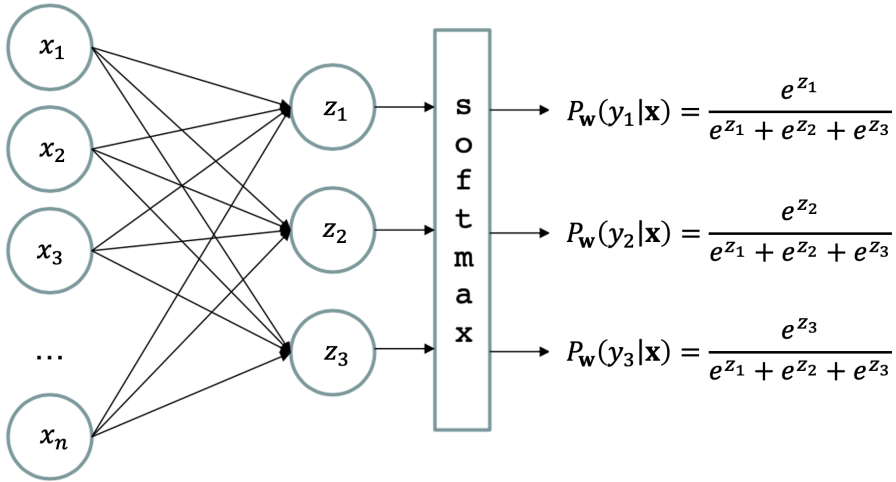
$$x_1 \; XOR \; x_2 = (x_1 \; AND \; \neg x_2) \; OR \; (\neg x_1 \; AND \; x_2)$$

Neural networks non-linearity



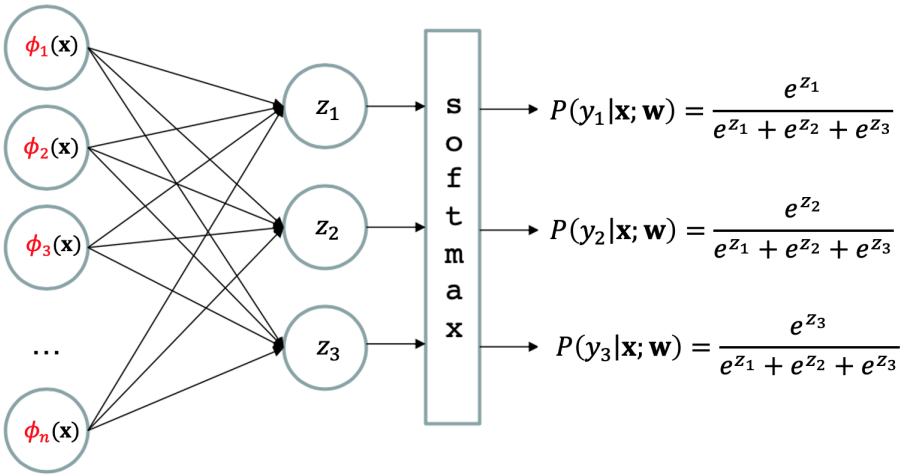
Multi-class logistic regression – a single layer neural network

special case of neural network



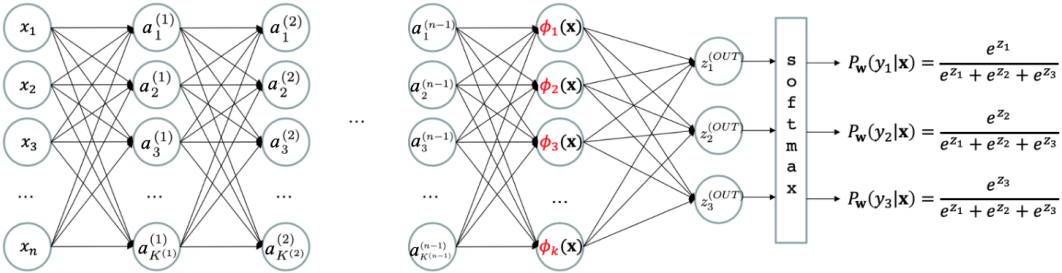
Multi-class logistic regression – a single layer neural network

special case of neural network



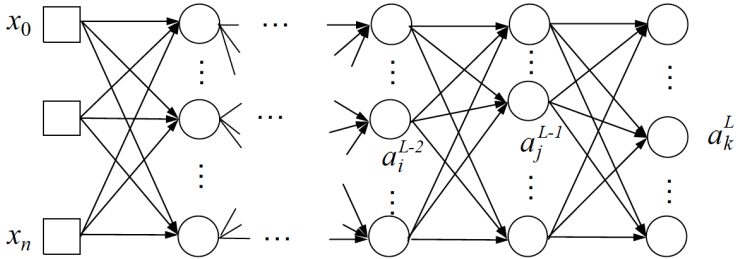
Deep Neural Network

Also learn the **features**!



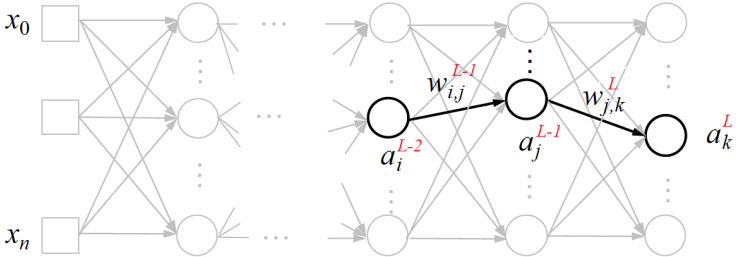
$$a_i^{(l)} = g\underbrace{\left(\sum_j w_{i,j}^{(l)} a_j^{(l-1)}\right)}_{z_i^{(l)}}; \quad g \text{ – non-linear activation function}$$

Backpropagation in neural nets: Derivation



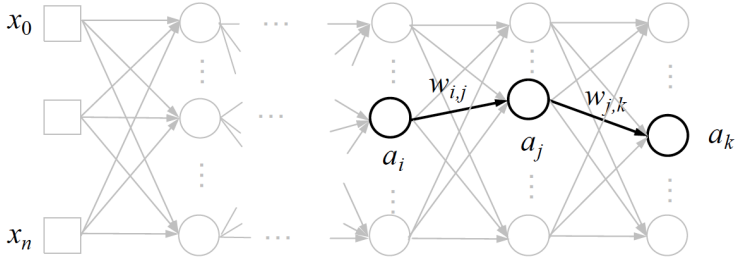
Suppose we have L layers. Denote the inputs by $x_0 \ldots x_n$. The outputs are $a_0^L \ldots a_n^L$, and a_j^{L-1} is the j-th activation at the layer L-1.

Backpropagation in neural nets: Derivation



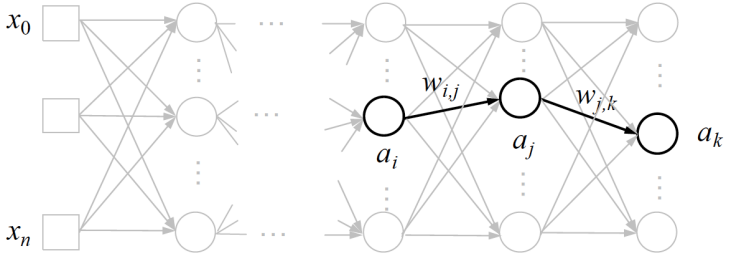
$w_{j,k}^{L}$ is the weight connecting a_j^{L-1} and a_k^{L}

Backpropagation in neural nets: Derivation



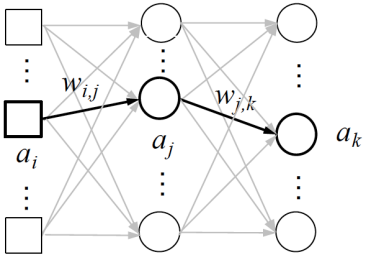
For compactness, we omit the superscript that denotes the level.

Backpropagation in neural nets: Derivation



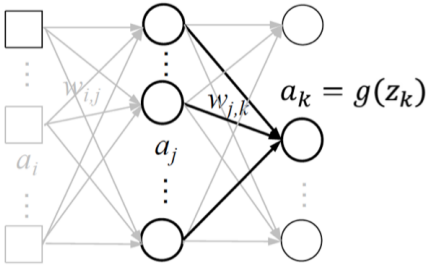
For compactness, we omit the superscript that denotes the level.
We also denote by a_k the output activations, by a_j the activations of the internal units and by a_i the "external" activations, i.e., the inputs x_i

Backpropagation in neural nets: Derivation



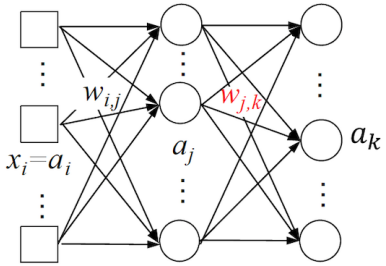
For compactness, we omit the superscript that denotes the level.
We also denote by a_k the output activations, by a_j the activations of the internal units and by a_i the "external" activations, i.e., the inputs x_i

Backpropagation in neural nets: Derivation



$$a_k = g(z_k) \qquad z_k = \sum_j w_{jk}\, a_j$$

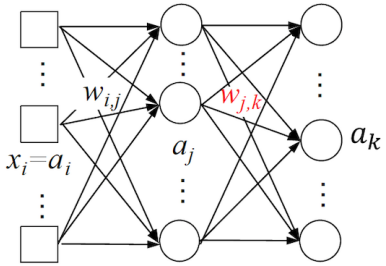
Backpropagation in neural nets: Derivation



$$z_k = \sum_j w_{j,k} a_j$$

$$a_k = g(z_k)$$

Backpropagation in neural nets: Derivation

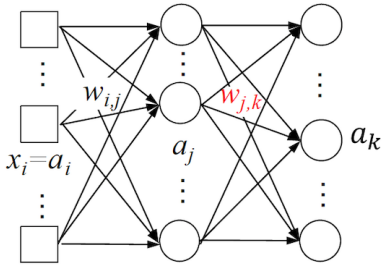


$$z_k = \sum_j w_{j,k} a_j$$

$$a_k = g(z_k)$$

$$w_{j,k} \leftarrow w_{j,k} - \alpha \frac{\partial Loss}{\partial w_{j,k}}$$

Backpropagation in neural nets: Derivation



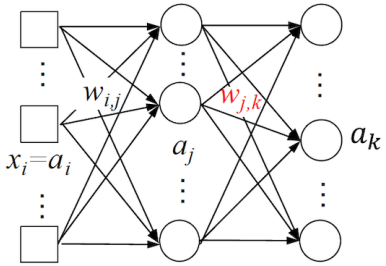
$$Loss = \sum_k (y_k - a_k)^2$$

$$z_k = \sum_j w_{j,k} a_j$$

$$a_k = g(z_k)$$

$$w_{j,k} \leftarrow w_{j,k} - \alpha \frac{\partial Loss}{\partial w_{j,k}}$$

Backpropagation in neural nets: Derivation



$$z_k = \sum_j w_{j,k} a_j$$

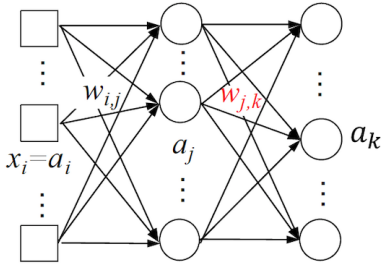
$$a_k = g(z_k)$$

$$w_{j,k} \leftarrow w_{j,k} - \alpha \frac{\partial Loss}{\partial w_{j,k}}$$

$$Loss = \sum_k (y_k - a_k)^2$$

$$\frac{\partial Loss}{\partial w_{j,k}} = \frac{\partial Loss}{\partial a_k} \frac{\partial a_k}{\partial z_k} \frac{\partial z_k}{\partial w_{j,k}}$$ (chain rule for derivatives)

Backpropagation in neural nets: Derivation



$$z_k = \sum_j w_{j,k} a_j$$

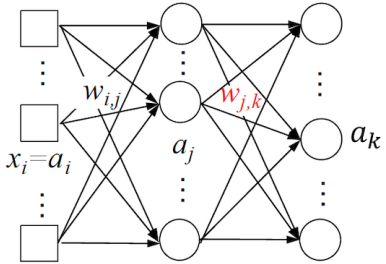
$$a_k = g(z_k)$$

$$w_{j,k} \leftarrow w_{j,k} - \alpha \frac{\partial Loss}{\partial w_{j,k}}$$

$$Loss = \sum_k (y_k - a_k)^2$$

$$\frac{\partial Loss}{\partial w_{j,k}} = \frac{\partial Loss}{\partial a_k} \frac{\partial a_k}{\partial z_k} \frac{\partial z_k}{\partial w_{j,k}} = \frac{\partial z_k}{\partial w_{j,k}} \frac{\partial a_k}{\partial z_k} \frac{\partial Loss}{\partial a_k}$$

Backpropagation in neural nets: Derivation



$$z_k = \sum_j w_{j,k} a_j$$

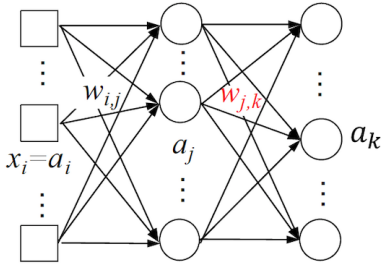
$$a_k = g(z_k)$$

$$w_{j,k} \leftarrow w_{j,k} - \alpha \frac{\partial Loss}{\partial w_{j,k}}$$

$$Loss = \sum_k (y_k - a_k)^2$$

$$\begin{aligned} \frac{\partial Loss}{\partial w_{j,k}} &= \frac{\partial Loss}{\partial a_k} \frac{\partial a_k}{\partial z_k} \frac{\partial z_k}{\partial w_{j,k}} = \frac{\partial z_k}{\partial w_{j,k}} \frac{\partial a_k}{\partial z_k} \frac{\partial Loss}{\partial a_k} \\ &= a_j g'(z_k)[-2(y_k - a_k)] \end{aligned}$$

Backpropagation in neural nets: Derivation



$$z_k = \sum_j w_{j,k} a_j$$

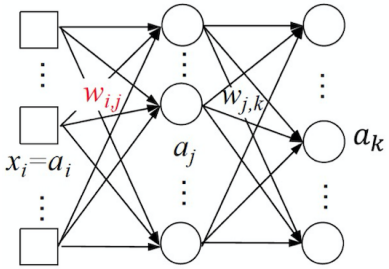
$$a_k = g(z_k)$$

$$w_{j,k} \leftarrow w_{j,k} - \alpha \frac{\partial Loss}{\partial w_{j,k}}$$

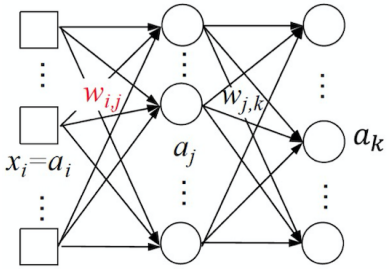
$$Loss = \sum_k (y_k - a_k)^2$$

$$\begin{aligned} \frac{\partial Loss}{\partial w_{j,k}} &= \frac{\partial Loss}{\partial a_k} \frac{\partial a_k}{\partial z_k} \frac{\partial z_k}{\partial w_{j,k}} = \frac{\partial z_k}{\partial w_{j,k}} \frac{\partial a_k}{\partial z_k} \frac{\partial Loss}{\partial a_k} \\ &= a_j g'(z_k)[-2(y_k - a_k)] \\ &= a_j \Delta_k \end{aligned}$$

Backpropagation in neural nets: Derivation



Backpropagation in neural nets: Derivation

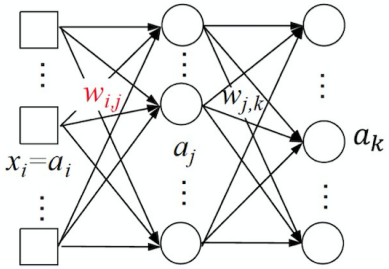


$$z_j = \sum_j w_{i,j} a_i$$

$$a_j = g(z_j)$$

$$w_{i,j} \leftarrow w_{i,j} - \alpha \frac{\partial Loss}{\partial w_{i,j}}$$

Backpropagation in neural nets: Derivation



$$z_j = \sum_j w_{i,j} a_i$$

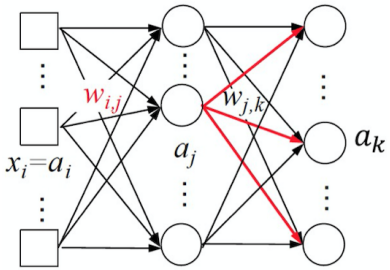
$$a_j = g(z_j)$$

$$w_{i,j} \leftarrow w_{i,j} - \alpha \frac{\partial Loss}{\partial w_{i,j}}$$

$$Loss = \sum_k (y_k - a_k)^2$$

$$\frac{\partial Loss}{\partial w_{j,k}} = \frac{\partial z_j}{\partial w_{i,j}} \frac{\partial a_j}{\partial z_j} \frac{\partial Loss}{\partial a_j}$$

Backpropagation in neural nets: Derivation



$$z_j = \sum_j w_{i,j} a_i$$

$$a_j = g(z_j)$$

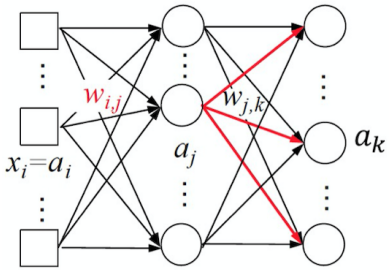
$$w_{i,j} \leftarrow w_{i,j} - \alpha \frac{\partial Loss}{\partial w_{i,j}}$$

$$Loss = \sum_k (y_k - a_k)^2$$

$$\frac{\partial Loss}{\partial w_{i,j}} = \frac{\partial z_j}{\partial w_{i,j}} \frac{\partial a_j}{\partial z_j} \frac{\partial Loss}{\partial a_j} = a_i g'(z_j) \sum_k \frac{\partial Loss}{\partial a_k} \frac{\partial a_k}{\partial a_j} = a_i g'(z_j) \sum_k \frac{\partial Loss}{\partial a_k} \frac{\partial a_k}{\partial z_k} \frac{\partial z_k}{\partial a_j}$$

$$= a_i g'(z_j) \sum_k \underbrace{\frac{\partial z_k}{\partial a_j}}_{w_{j,k}} \underbrace{\frac{\partial a_k}{\partial z_k}}_{g'(z_k)} \frac{\partial Loss}{\partial a_k}$$

Backpropagation in neural nets: Derivation



$$z_j = \sum_j w_{i,j} a_i$$

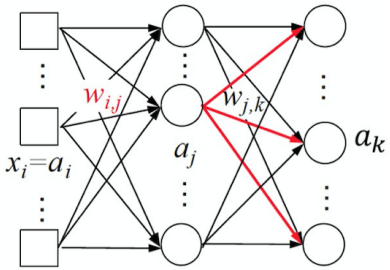
$$a_j = g(z_j)$$

$$w_{i,j} \leftarrow w_{i,j} - \alpha \frac{\partial Loss}{\partial w_{i,j}}$$

$$Loss = \sum_k (y_k - a_k)^2$$

$$\frac{\partial Loss}{\partial w_{i,j}} = \frac{\partial z_j}{\partial w_{i,j}} \frac{\partial a_j}{\partial z_j} \frac{\partial Loss}{\partial a_j} = a_i g'(z_j) \sum_k w_{j,k} \underbrace{g'(z_k) \frac{\partial Loss}{\partial a_k}}_{\Delta_k}$$

Backpropagation in neural nets: Derivation



$$z_j = \sum_j w_{i,j} a_i$$

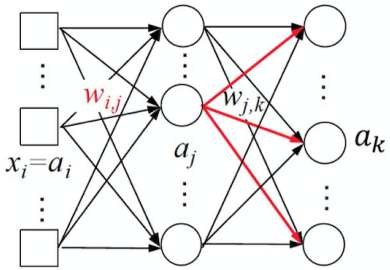
$$a_j = g(z_j)$$

$$w_{i,j} \leftarrow w_{i,j} - \alpha \frac{\partial Loss}{\partial w_{i,j}}$$

$$Loss = \sum_k (y_k - a_k)^2$$

$$\frac{\partial Loss}{\partial w_{i,j}} = \frac{\partial z_j}{\partial w_{i,j}} \frac{\partial a_j}{\partial z_j} \frac{\partial Loss}{\partial a_j} = a_i g'(z_j) \sum_k w_{j,k} \underbrace{g'(z_k) \frac{\partial Loss}{\partial a_k}}_{\Delta_k} = a_i \Delta_j$$

Backpropagation in neural nets: Derivation



$$z_j = \sum_j w_{i,j} a_i$$

$$a_j = g(z_j)$$

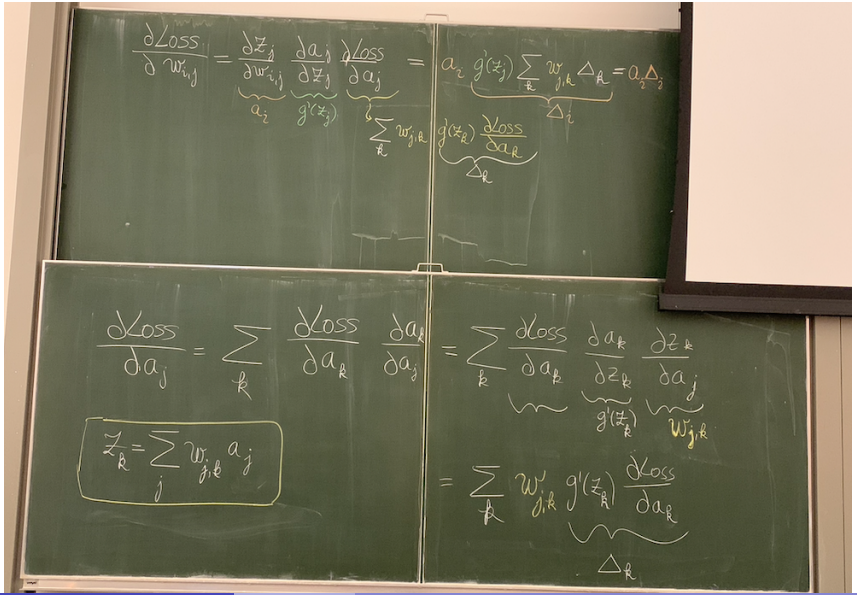
$$w_{i,j} \leftarrow w_{i,j} - \alpha \frac{\partial Loss}{\partial w_{i,j}}$$

$$Loss = \sum_k (y_k - a_k)^2$$

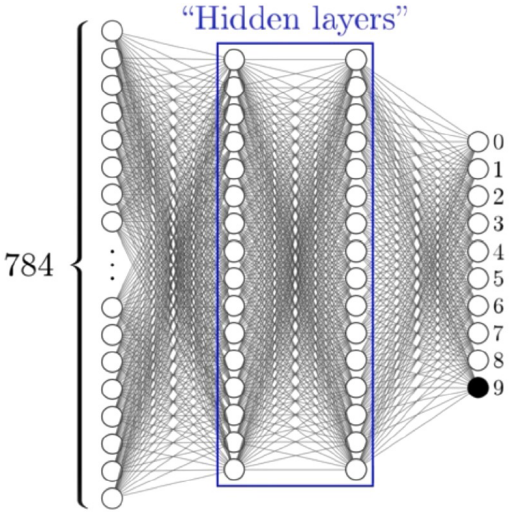
$$\frac{\partial Loss}{\partial w_{i,j}} = \frac{\partial z_j}{\partial w_{i,j}} \frac{\partial a_j}{\partial z_j} \frac{\partial Loss}{\partial a_j} = a_i g'(z_j) \sum_k w_{j,k} \underbrace{g'(z_k) \frac{\partial Loss}{\partial a_k}}_{\Delta_k} = a_i \Delta_j$$

$$\Delta_j = g'(z_j) \sum_k w_{j,k} \Delta_k$$

Backpropagation in neural nets: Derivation

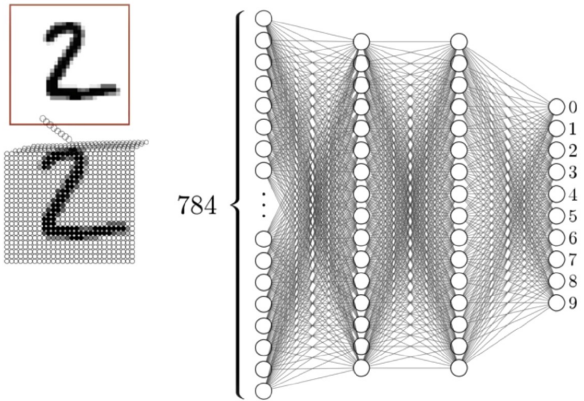


The multilayer perceptron



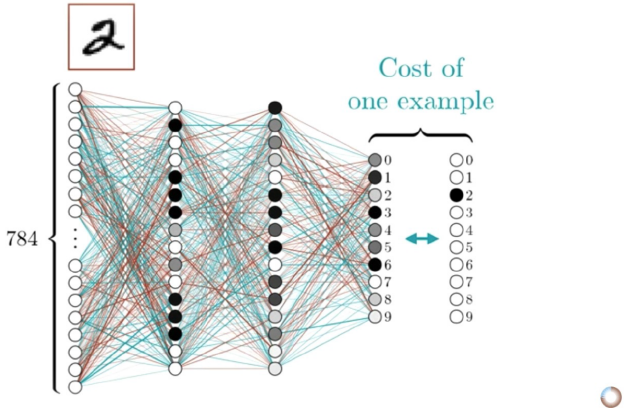
3Blue1Brown math channel `https://www.youtube.com/watch?v=Ilg3gGewQ5U`

The multilayer perceptron



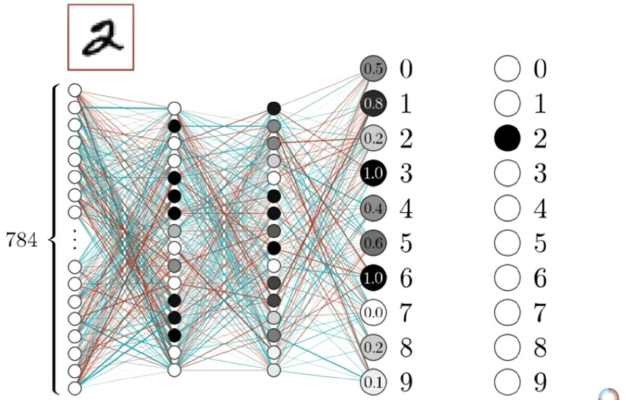
3Blue1Brown math channel https://www.youtube.com/watch?v=Ilg3gGewQ5U

The multilayer perceptron



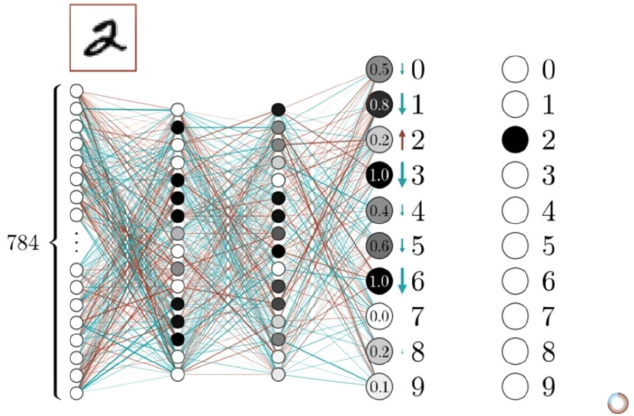
3Blue1Brown math channel https://www.youtube.com/watch?v=Ilg3gGewQ5U

The multilayer perceptron



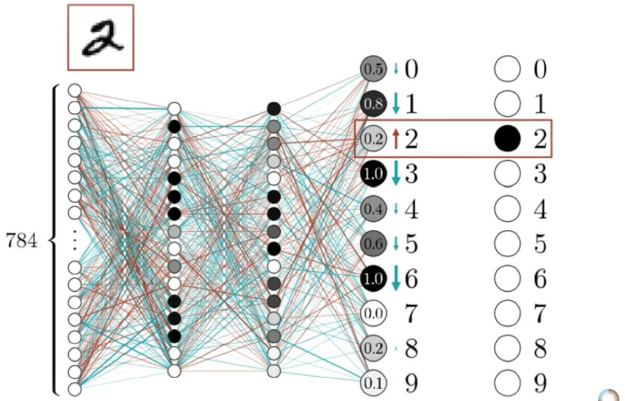
3Blue1Brown math channel `https://www.youtube.com/watch?v=Ilg3gGewQ5U`

The multilayer perceptron



3Blue1Brown math channel `https://www.youtube.com/watch?v=Ilg3gGewQ5U`

The multilayer perceptron



3Blue1Brown math channel `https://www.youtube.com/watch?v=aircAruvnKk`

Training deep neural networks

In deep learning we commonly use the negative log likelihood as the loss function
Thus the same as ML learning for logistic regression:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} -\ell(\mathbf{w}) = \arg\min_{\mathbf{w}} -\prod_{i=1}^{N} \log P(y^{(i)}|\mathbf{x}^{(i)}, \mathbf{w})$$

just $\mathbf{w}$ tends to be a much, much larger vector

Run gradient descent
and stop when the loss function on hold-out data starts to increase
(i.e., when log likelihood of hold-out data starts to decrease)

How to compute all the derivatives?

- Derivatives can be computed by following well-defined procedures
- Apply chain rule:
  - If $f(x) = g(h(x))$
    Then $f'(x) = g'(h(x))h'(x)$
- Automatic differentiation software
  - E.g. Theano, TensorFlow, PyTorch, Chainer
  - Only need to program the function $g(\mathbf{x}, \mathbf{y}, \mathbf{w})$
  - Can automatically compute all derivatives w.r.t. all entries in $\mathbf{w}$
  - This is typically done by caching info during forward computation pass, and then doing a backward pass = "backpropagation"
  - Autodiff / Backpropagation can often be done at computational cost comparable to the forward pass

Universal Function Approximation Theorem

Multilayer Feedforward Networks are Universal Approximators

KURT HORNIK

Technische Universität Wien

MAXWELL STINCHCOMBE AND HALBERT WHITE

University of California, San Diego

(*Received* 16 *September* 1988; *revised and accepted* 9 *March* 1989)

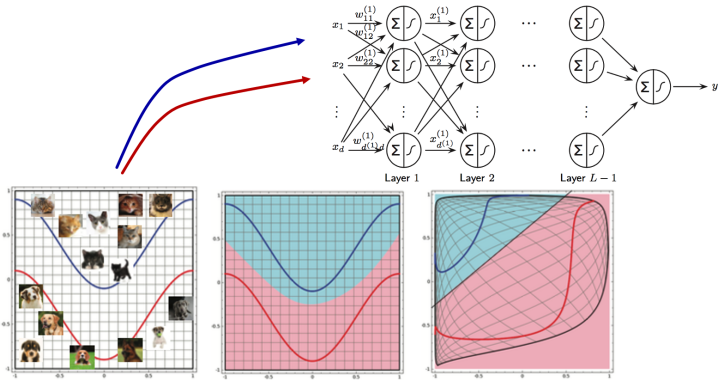
Abstract—*This paper rigorously establishes that standard multilayer feedforward networks with as few as one hidden layer using arbitrary squashing functions are capable of approximating any Borel measurable function from one finite dimensional space to another to any desired degree of accuracy, provided sufficiently many hidden units are available. In this sense, multilayer feedforward networks are a class of universal approximators.*

Hornik et al, Neural Networks, 1989

Given any continuous function $f(x)$, if a 2-layer neural network has enough hidden units, then there is a choice of weights that allow it to closely approximate $f(x)$.

Cybenko (1989) "Approximations by superpositions of sigmoidal functions"
Hornik (1991) "Approximation Capabilities of Multilayer Feedforward Networks"
Leshno and Schocken (1991) "Multilayer Feedforward Networks with Non-Polynomial Activation Functions Can Approximate Any Function"

Hidden layers and nonlinearities



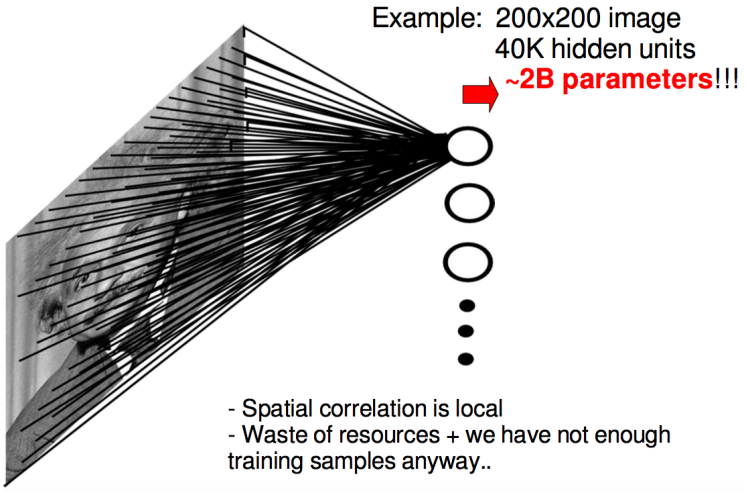
A multi-layer neural net can distort the input space to make the classes of data linearly separable

With more hidden layers this becomes more prominent and typically fewer hidden units needed then with shallow nets

Fun Neural Net Demo Site

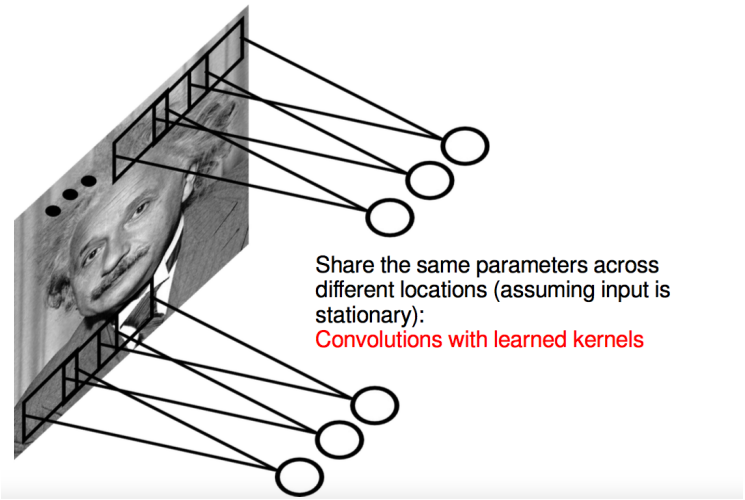
- Demo-site:
  - `http://playground.tensorflow.org/`

Convolutional Neural Networks (CNN) - idea and motivation



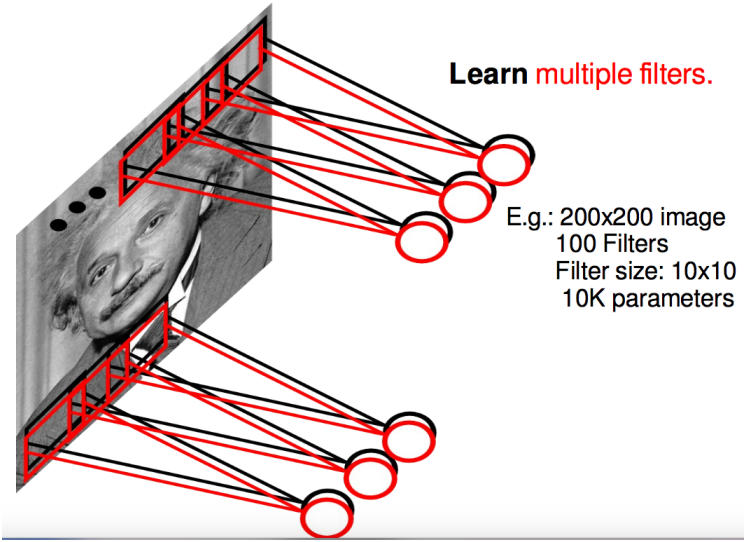
M. Ranzato (Facebook A.I. research): Image Classification with Deep Learning, 2015.

Convolutional Neural Networks (CNN) - idea and motivation



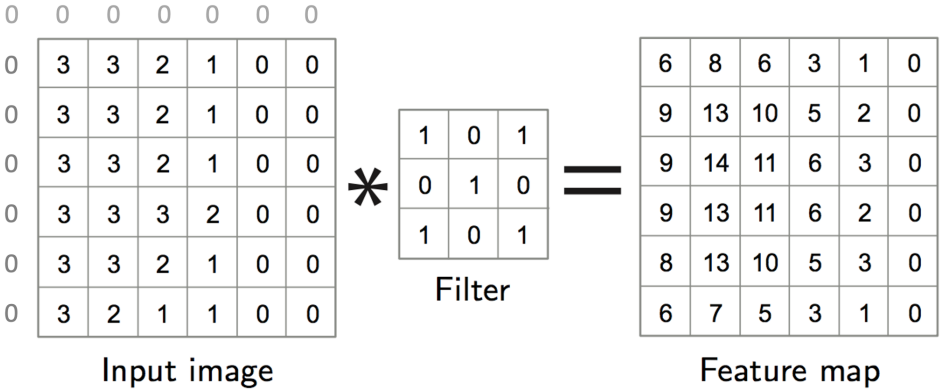
M. Ranzato (Facebook A.I. research): Image Classification with Deep Learning, 2015.

Convolutional Neural Networks (CNN) - idea and motivation

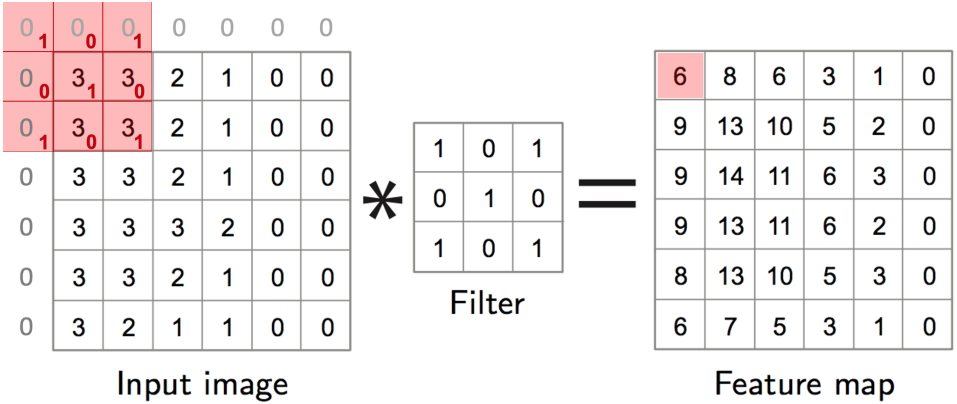


M. Ranzato (Facebook A.I. research): Image Classification with Deep Learning, 2015.

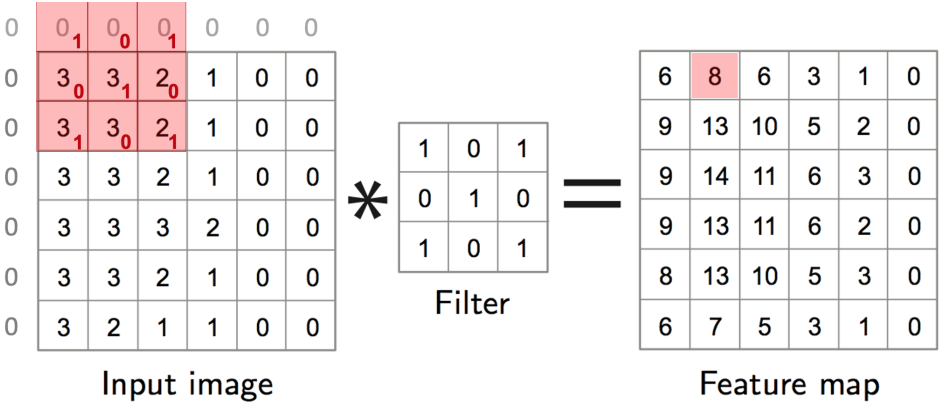
Convolutional layer



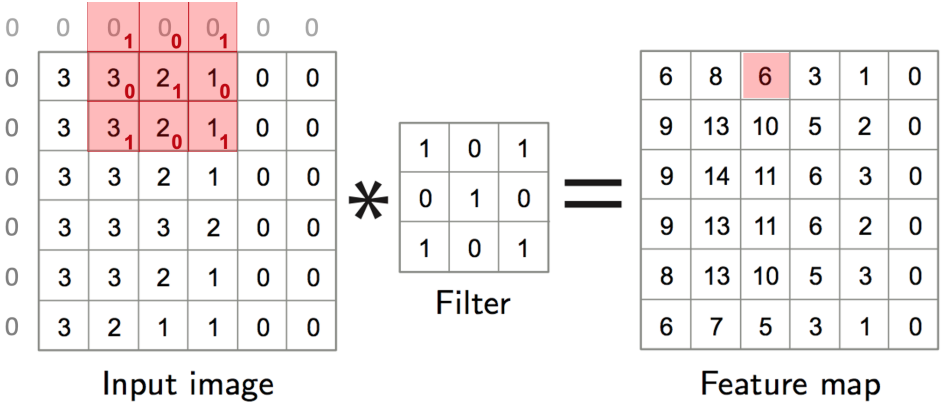
Convolutional layer



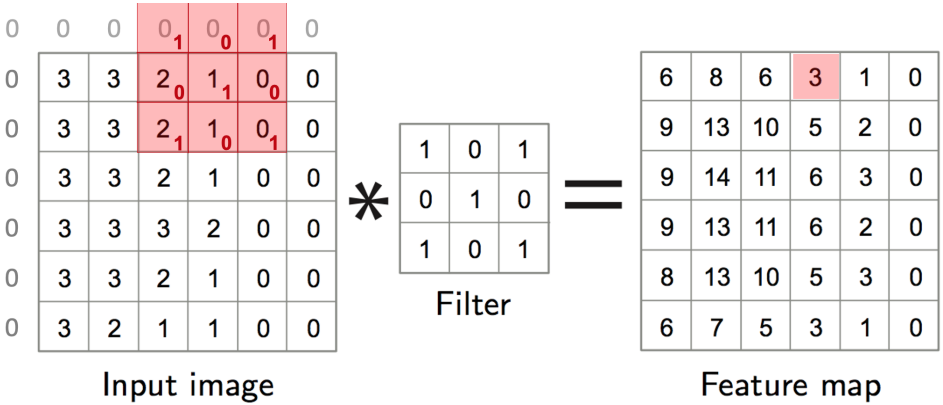
Convolutional layer



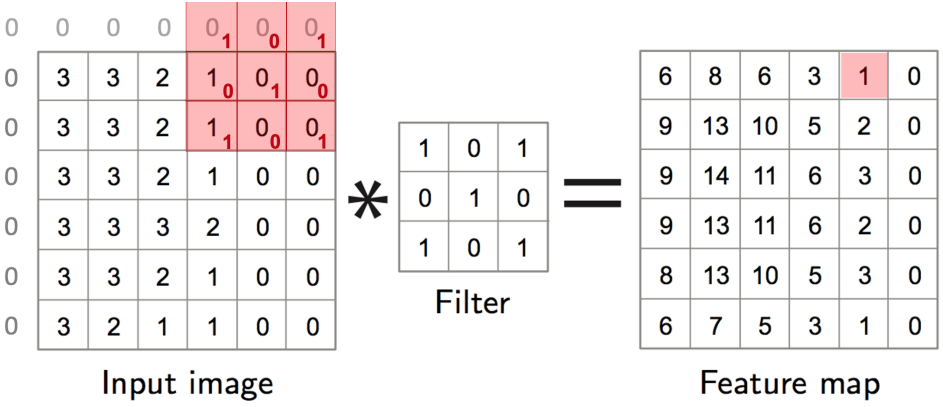
Convolutional layer



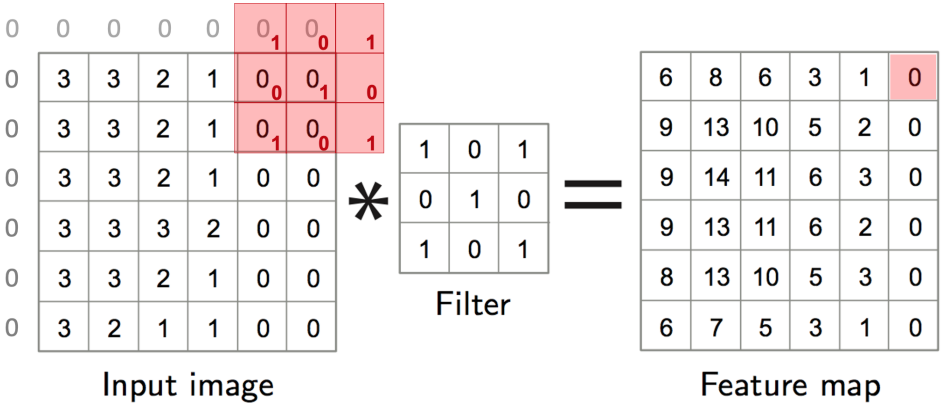
Convolutional layer



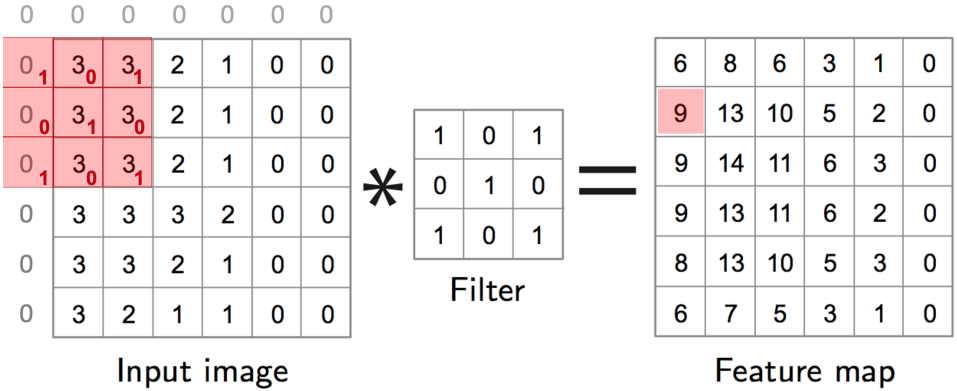
Convolutional layer



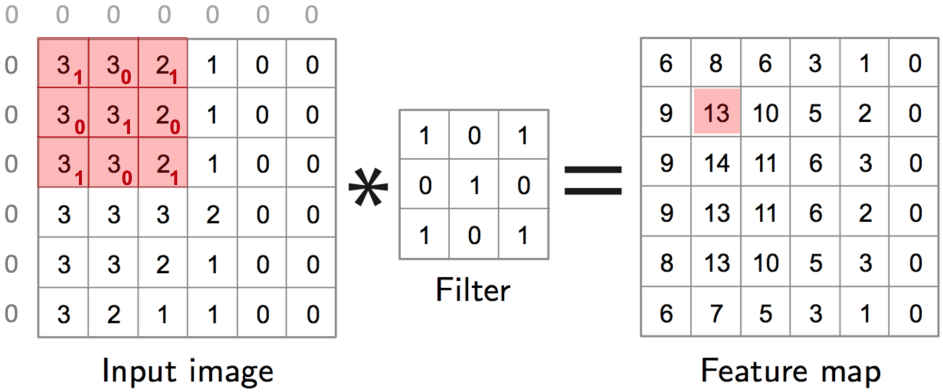
Convolutional layer



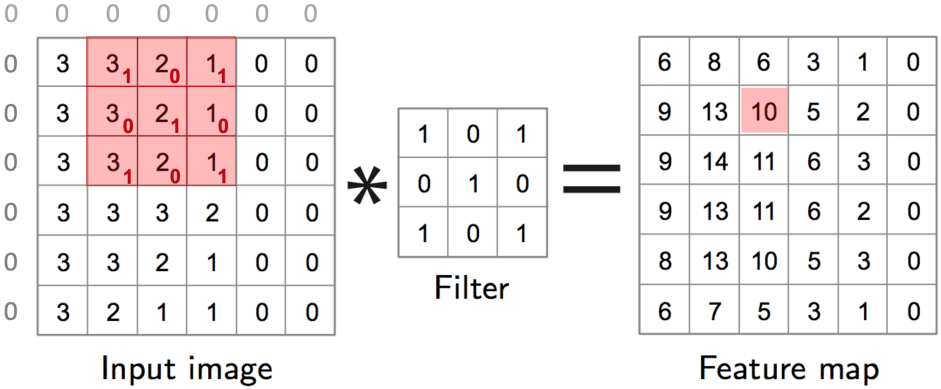
Convolutional layer



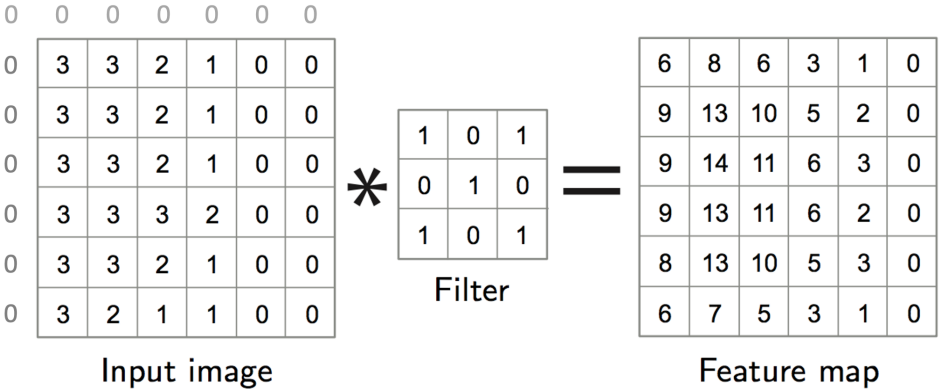
Convolutional layer



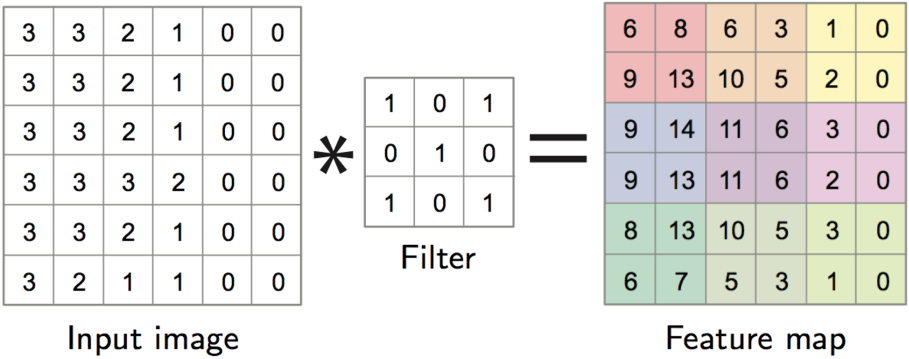
Convolutional layer



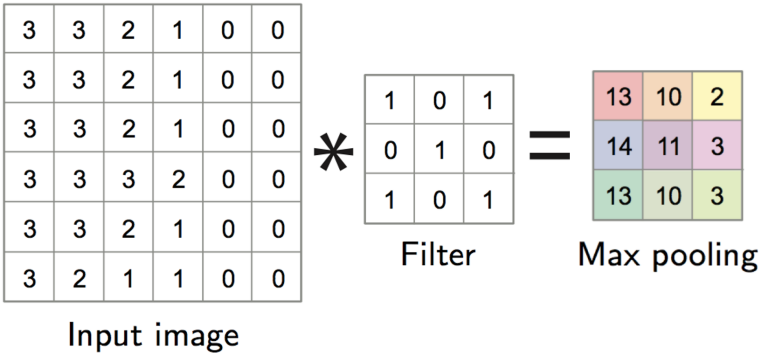
Convolutional layer



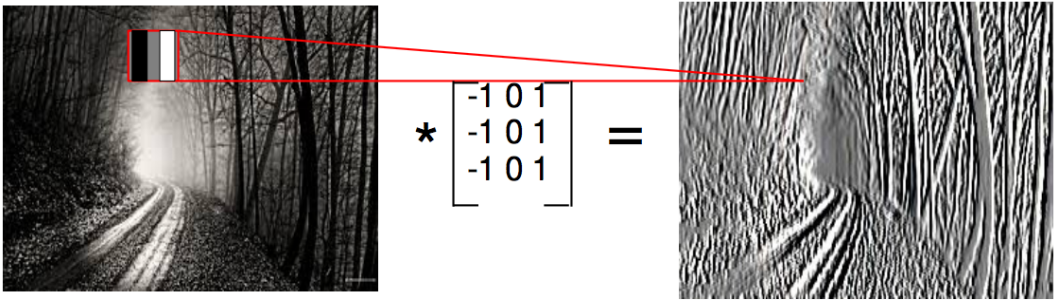
Convolutional layer



Convolutional layer

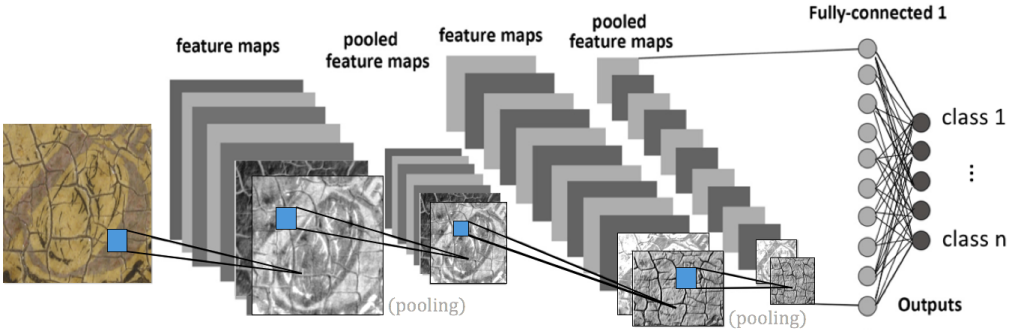


Convolutional layer



M. Ranzato (Facebook A.I. research): Image Classification with Deep Learning, 2015.

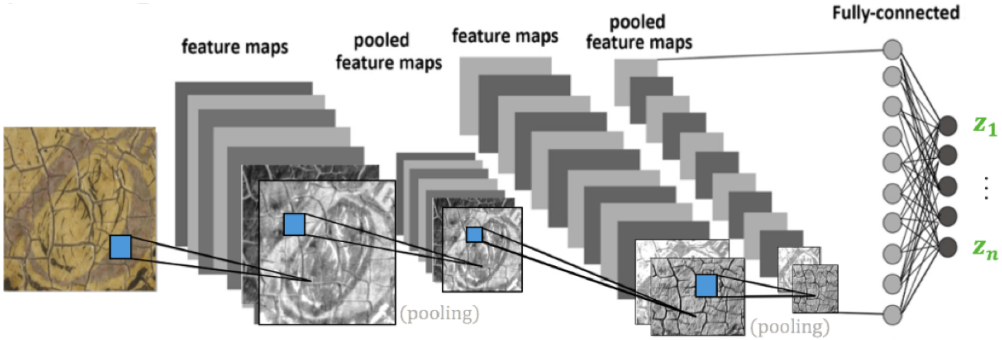
Convolutional Neural Networks (CNN)



Output at location (i, j) of the k-th feature map in the l-th layer:

$$a_{i,j}^{l,k} = g\Big(\sum_{m=1}^{M} \sum_{p=0}^{H_l-1} \sum_{q=0}^{W_l-1} w_{p,q}^{l,k,m} a_{(i+p),(j+q)}^{(l-1),m} + b^{l,k}\Big)$$

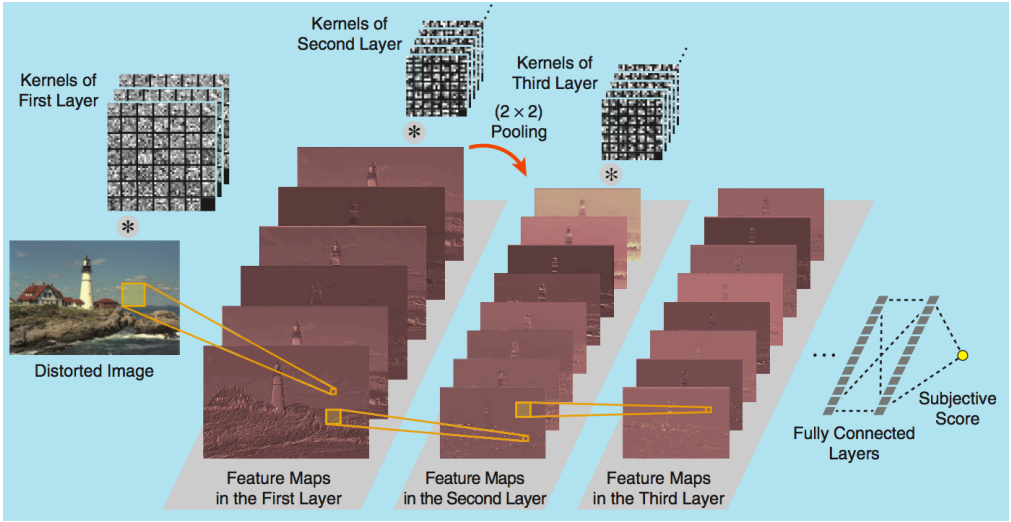
Convolutional Neural Networks (CNN)



Predicted probabilities of class labels using the softmax rule:

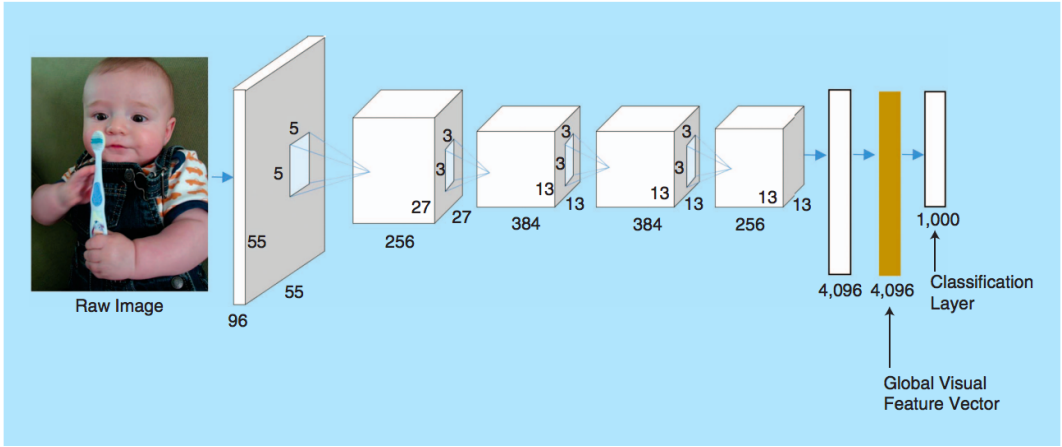
$$P(\text{class}(x_{i,j}) = c) = \frac{e^{z_c}}{\sum_k e^{z_k}}$$

CNN: Kernels and feature maps



J. Kim et al. Deep Convolutional Neural Models for Picture-Quality Prediction. IEEE Signal Processing Magazine, Nov. 2017.

A common representation of deep CNN architectures



X. He and L. Deng. Deep Learning for Image-to-Text Generation. IEEE Signal Processing Magazine, Nov. 2017.