Lecture 11. Object Recognition III
Deep Learning basics

703142. Computer Vision

Assoz.Prof. Antonio Rodríguez-Sánchez, PhD.

Tentative schedule

Date	Topic
06-03-2021	Introduction.
13-03-2021	Review: Image Formation. Introduction to OpenCV.
20-03-2021	Review: Image processing.
27-03-2021	Review: Feature extraction.
17-04-2021	Motion I: Optical Flow. OpenCV.
24-04-2021	Motion II: Spatiotemporal filters. Spatiotemporal analysis.
08-05-2021	Stereopsis I: Correspondence. RANSAC.
15-05-2021	Stereopsis II: Reconstruction. Rendering.
22-05-2021	Object Recognition I: Categories. Photo editing.
05-06-2021	Object Recognition II: Objects, faces, instances. Viola and Jones
12-06-2021	Object Recognition III: Deep Learning. CNNs in Pytorch.
19-06-2021	Computational Neuroscience.Exam questions.
26-06-2021	FINAL EXAM.

Neural networks

Feed-forward networks

Training

Backpropagation

Deep Learning

Neural networks

- Fix the number of basis functions in advance and allow them to be adaptive
- The parameter values are adapted during training
- In many cases they are more compact and faster to evaluate than a SVM
- The likelihood function is no longer a convex function of the model parameters
- McCulloh and Pitts (1943); Widrow and Hoff (1960); Rosenblatt (1962); Rumelhart et al. (1986)

Neural networks

- Determine the network parameters with a maximum likelihood framework
  - This involves the solution of a nonlinear optimization problem
  - Thus, evaluate the derivates of the log likelihood function with respect to the network parameters
  - This can be done efficiently by means of *error backpropagation*
  - Or approaching the problem from a Bayesian perspective

Neural networks

Feed-forward networks

Training

Backpropagation

Deep Learning

Feed-forward network functions

- Linear models for regression and classification are based on linear combination of fixed nonlinear basis functions

$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x})\right)$$

  - f is the activation function
  - Each basis function is a nonlinear function of a linear combination of the inputs
    - The coefficients in the linear combination are adaptive parameters

Feed-forward network functions

- Linear models for regression and classification are based on linear combination of fixed nonlinear basis functions

$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x})\right)$$

  - f is the activation function
  - Each basis function is a nonlinear function of a linear combination of the inputs
    - The coefficients in the linear combination are adaptive parameters
- We want to make the basis functions depend on parameters
  - And adjust those parameters along with the weights during training
- The basic neural network is a series of functional transformations

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

Feed-forward network functions

- The basic neural network is a series of functional transformations

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

  - a_j are the *activations*
  - And the activation function is $z_j = h(a_j)$
  - This corresponds to the outputs of the basis functions
    - These are the *hidden units*
    - Usually these functions are sigmoidal, e.g. tanh

Feed-forward network functions

- The basic neural network is a series of functional transformations

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

  - a_j are the *activations*
  - And the activation function is $z_j = h(a_j)$
  - This corresponds to the outputs of the basis functions
    - These are the *hidden units*
    - Usually these functions are sigmoidal, e.g. $\tanh$
  - These values are again combined to give *output unit activations*

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}$$

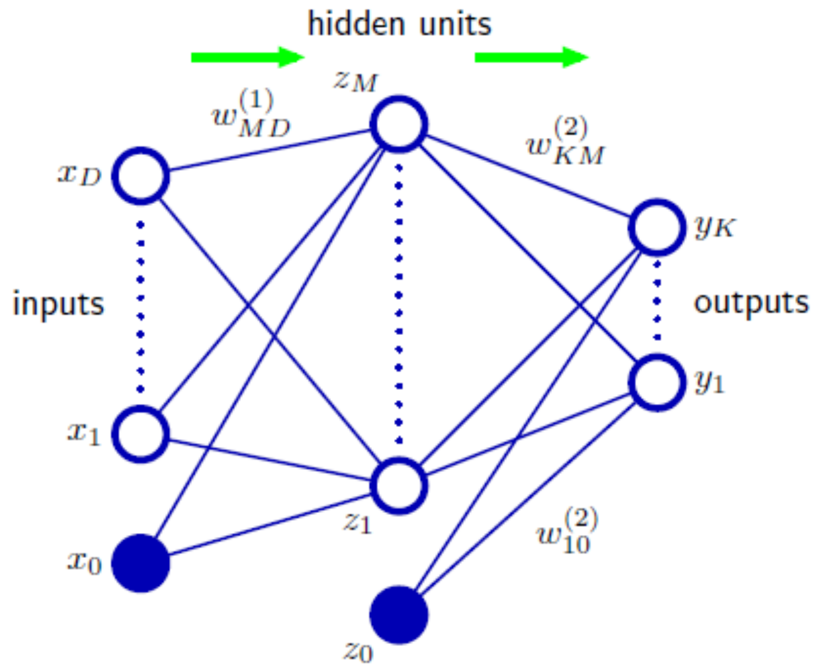
Feed-forward network functions

- The basic neural network is a series of functional transformations

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

  - a_j are the *activations*
  - And the activation function is $z_j = h(a_j)$
  - This corresponds to the outputs of the basis functions
    - These are the *hidden units*
    - Usually these functions are sigmoidal, e.g. tanh
  - These values are again combined to give *output unit activations*

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}$$



$$y_k = \sigma(a_k)$$

$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

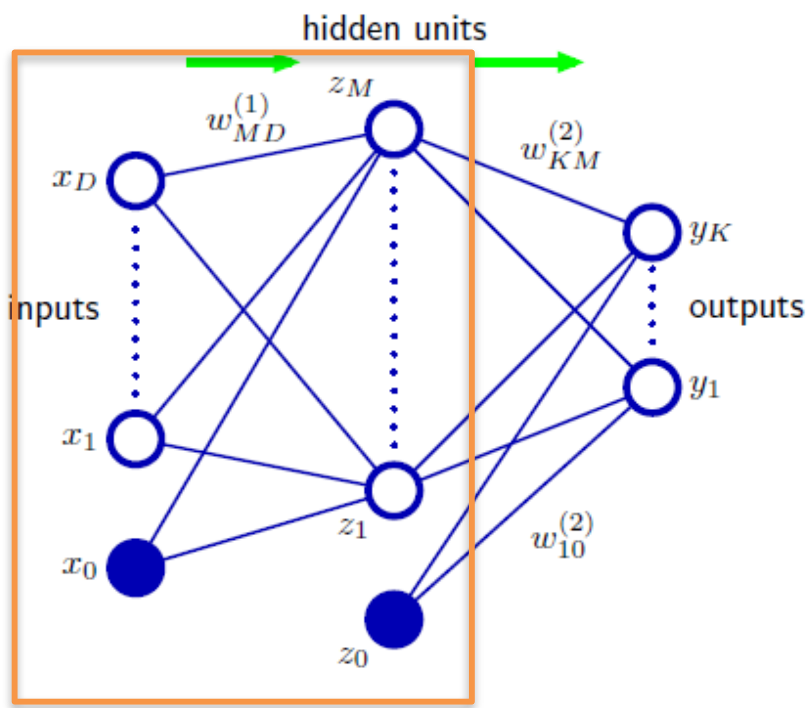
Feed-forward network functions

- The basic neural network is a series of functional transformations

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

  - a_j are the *activations*
  - And the activation function is $z_j = h(a_j)$
  - This corresponds to the outputs of the basis functions
    - These are the *hidden units*
    - Usually these functions are sigmoidal, e.g. tanh
  - These values are again combined to give *output unit activations*

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}$$



$$y_k = \sigma(a_k)$$

$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

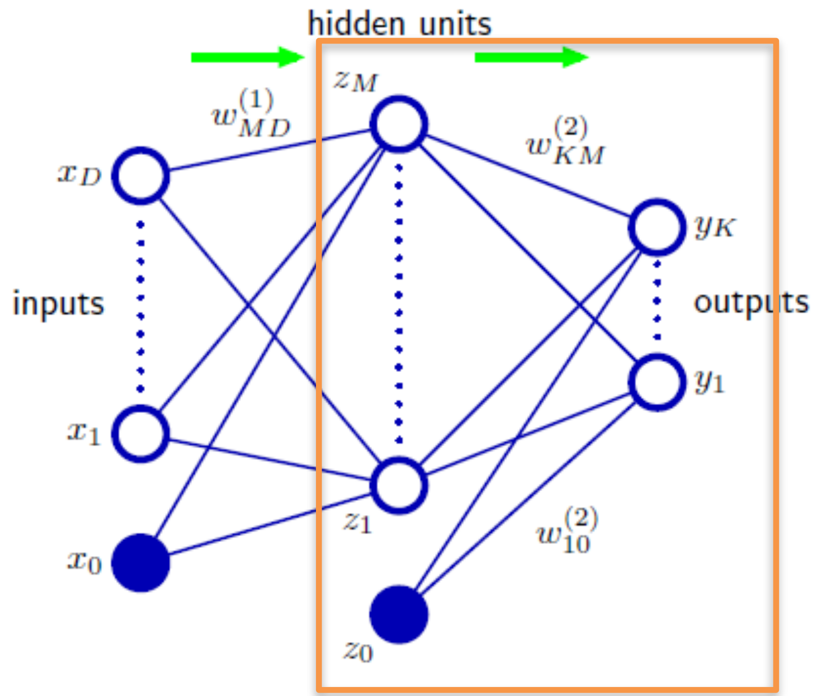
Feed-forward network functions

- The basic neural network is a series of functional transformations

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

  - a_j are the *activations*
  - And the activation function is $z_j = h(a_j)$
  - This corresponds to the outputs of the basis functions
    - These are the *hidden units*
    - Usually these functions are sigmoidal, e.g. tanh
  - These values are again combined to give *output unit activations*

$$\boxed{a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}}$$



$$y_k = \sigma(a_k)$$

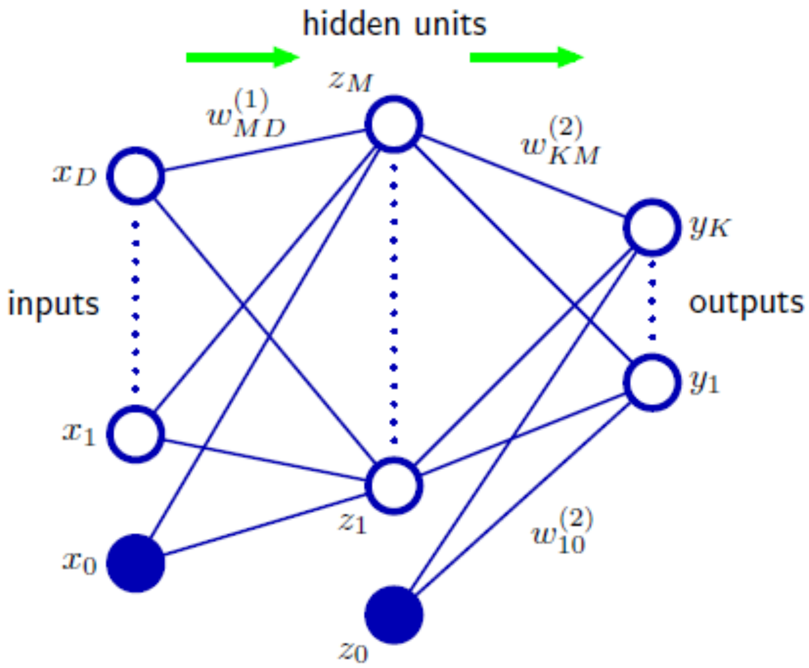
$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

Feed-forward network functions

- The basic neural network is a series of functional transformations

$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x})\right)$$

$$= \boxed{\sigma}\left(\sum_{j=1}^{M} w_{kj}^{(2)} \boxed{h}\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$



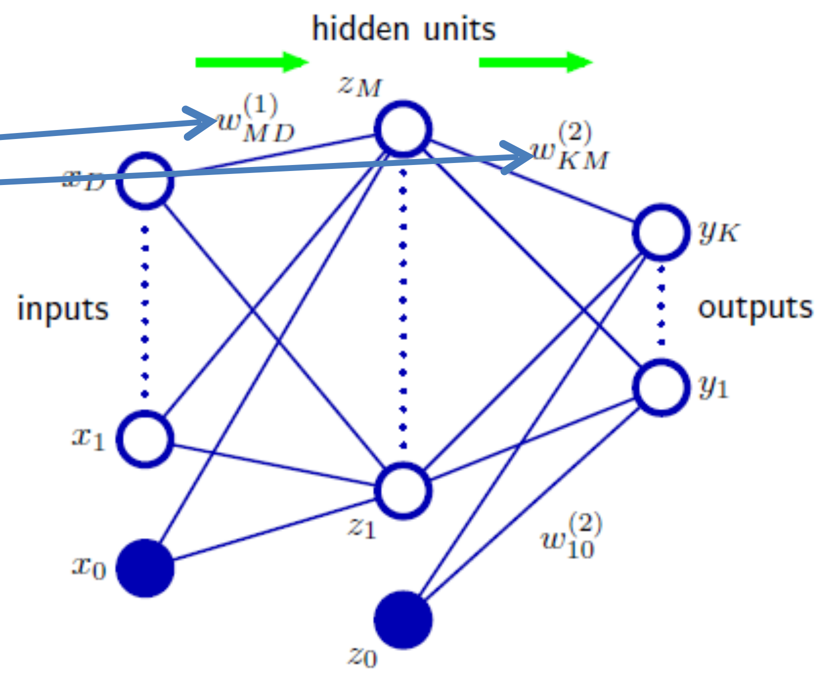
$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

Feed-forward network functions

- The basic neural network is a series of functional transformations

$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x})\right)$$

$$= \boxed{\sigma}\left(\sum_{j=1}^{M} w_{kj}^{(2)} \boxed{h}\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$



  - Thus a neural network is a nonlinear function from a set of input variables $\{x_i\}$ to a set of output variables $\{y_i\}$ by a vector w of adjustable parameters

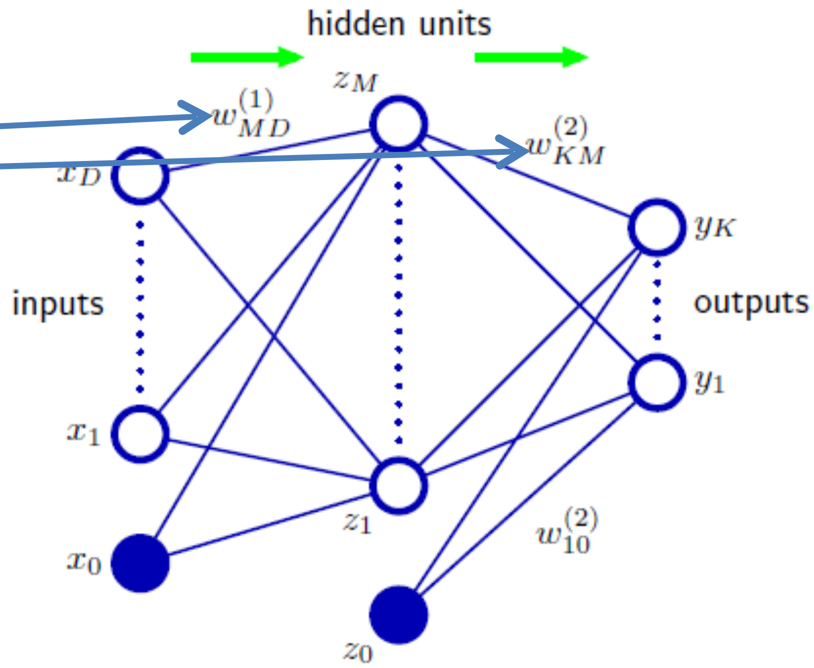
$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

Feed-forward network functions

- The basic neural network is a series of functional transformations

$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x})\right)$$

$$= \boxed{\sigma}\left(\sum_{j=1}^{M} w_{kj}^{(2)} \boxed{h}\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$



  - Thus a neural network is a nonlinear function from a set of input variables $\{x_i\}$ to a set of output variables $\{y_i\}$ by a vector $\mathbf{w}$ of adjustable parameters
- This is called *forward propagation*

$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

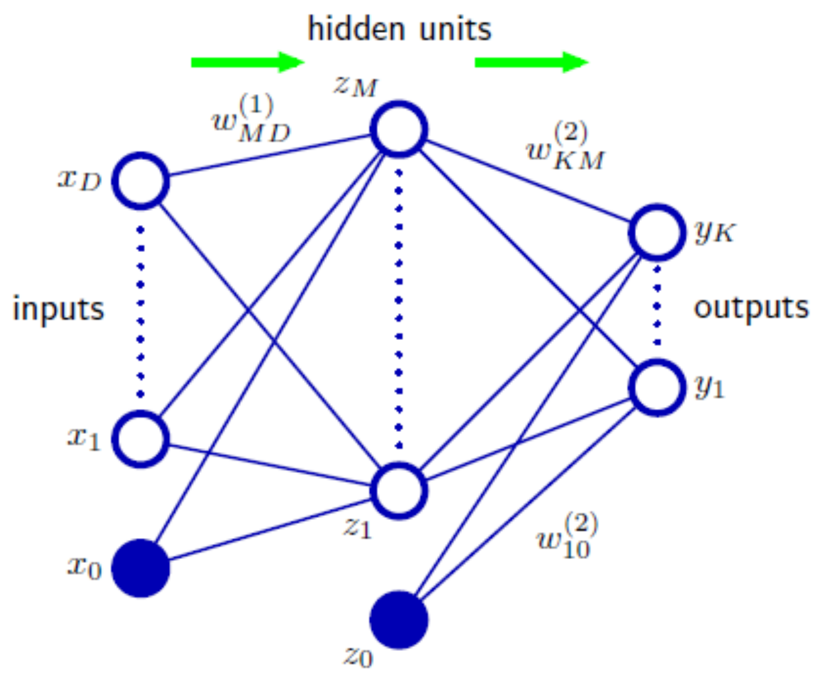
Feed-forward network functions

- The basic neural network is a series of functional transformations

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i$$

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma \left(\sum_{j=0}^{M} w_{kj}^{(2)} h \left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i \right) \right)$$

  - This type of neural network is called the *multilayer perceptron*
  - Usually 3 layers
    - But we can add more hidden layers

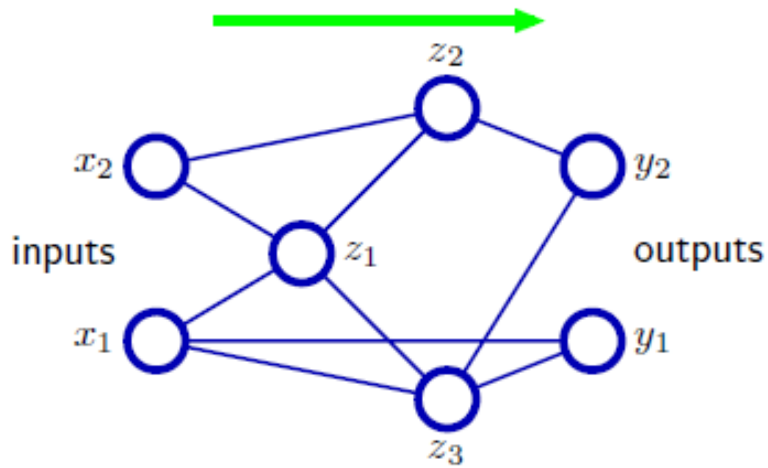


Feed-forward network functions

- The basic neural network is a series of functional transformations

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i$$

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right)$$



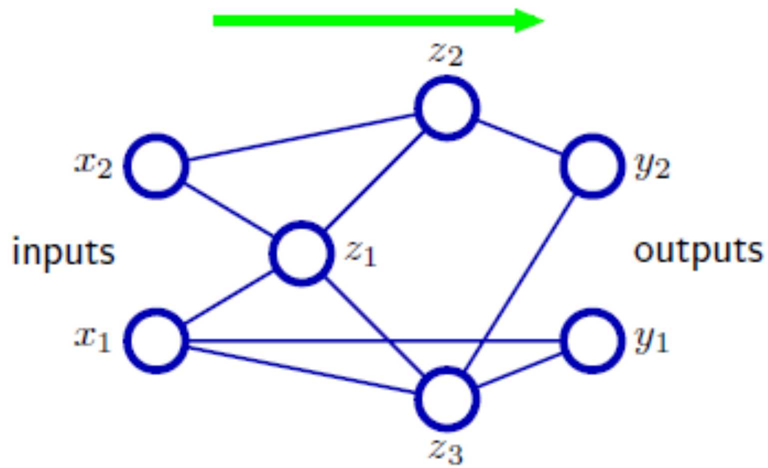
  - This type of neural network is called the *multilayer perceptron*
  - Usually 3 layers
    - But we can add more hidden layers

Feed-forward network functions

- The basic neural network is a series of functional transformations

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i$$

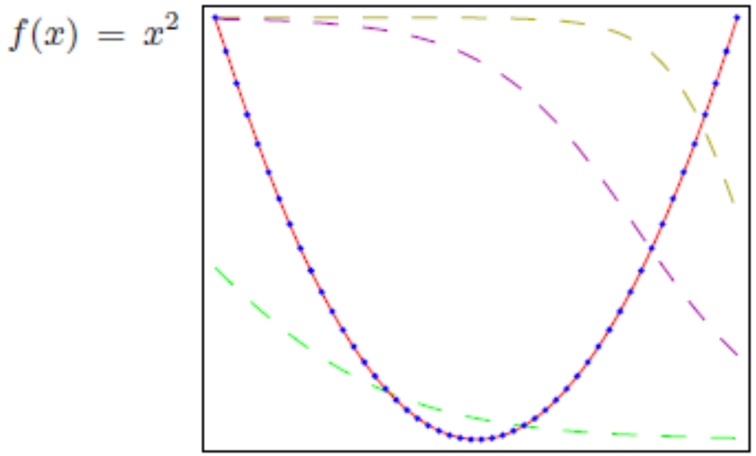
$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right)$$



  - This type of neural network is called the *multilayer perceptron*
  - Usually 3 layers
    - But we can add more hidden layers
- Neural networks are *Universal approximators*
  - The key problem is to find suitable parameter values given a set of training data

Feed-forward network functions

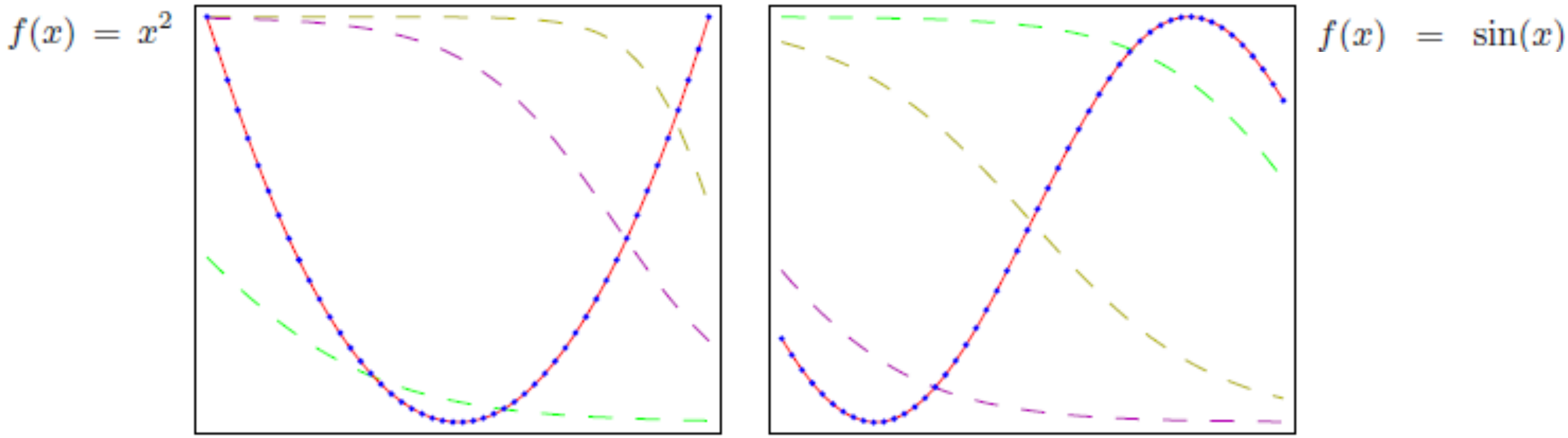
- Universal approximators



(a)

Feed-forward network functions

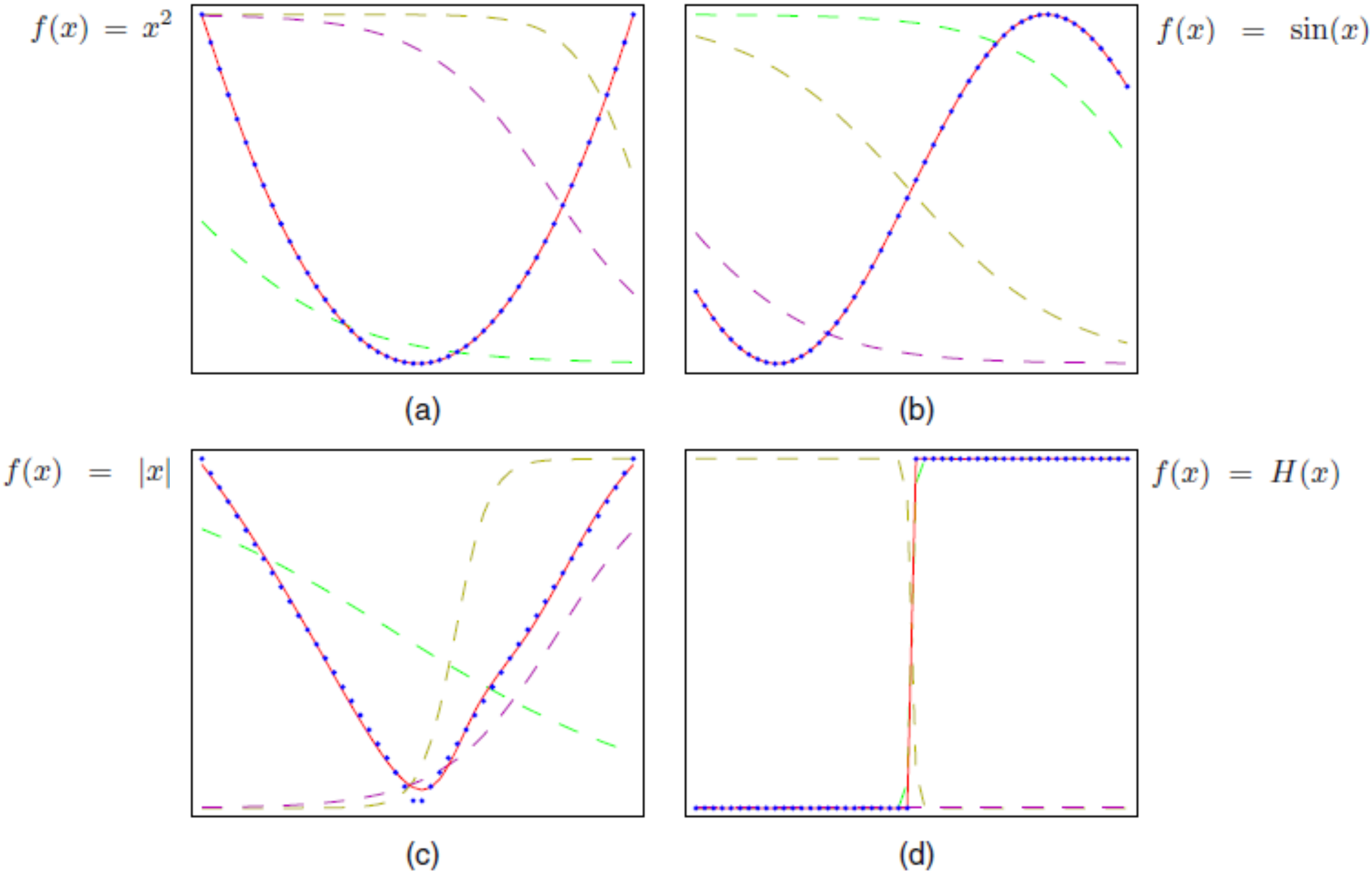
- Universal approximators



(a) (b)

Feed-forward network functions

- Universal approximators



Neural networks

Feed-forward networks

Training

Backpropagation

Deep Learning

Training

- Depends on the task
- **For regression**: Linear outputs and a minimize sum-of-squares function

Training

- Depends on the task
- **For regression**: Linear outputs and a minimize sum-of-squares function
- Output is $p(t|\mathbf{x}, \mathbf{w}) = \mathcal{N}\left(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}\right)$
  – The likelihood and its (negative) logarithmic version (error function)

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} p(t_n|\mathbf{x}_n, \mathbf{w}, \beta) \qquad \frac{\beta}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 - \frac{N}{2}\ln\beta + \frac{N}{2}\ln(2\pi)$$

Training

- Depends on the task
- **For regression**: Linear outputs and a minimize sum-of-squares function
- Output is $p(t|\mathbf{x}, \mathbf{w}) = \mathcal{N}\left(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}\right)$
  - The likelihood and its (negative) logarithmic version (error function)

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} p(t_n|\mathbf{x}_n, \mathbf{w}, \beta)$$

$$\frac{\beta}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 - \frac{N}{2}\ln\beta + \frac{N}{2}\ln(2\pi)$$

  - Learn the parameters (maximizing log likelihood as usual)

$$E(\boxed{\mathbf{w}}) = \frac{1}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2$$

$$\frac{1}{\boxed{\beta_{\mathrm{ML}}}} = \frac{1}{N}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}_{\mathrm{ML}}) - t_n\}^2$$

Training

- Depends on the task
- **For binary classification**: We use a logistic sigmoid outputs and a cross-entropy error function

Training

- Depends on the task
- **For binary classification**: We use a logistic sigmoid outputs and a cross-entropy error function
- Output is $y = \sigma(a) \equiv \dfrac{1}{1 + \exp(-a)}$
  - The conditional distribution is given by a Bernoulli distribution

$$p(t|\mathbf{x}, \mathbf{w}) = y(\mathbf{x}, \mathbf{w})^t \left\{1 - y(\mathbf{x}, \mathbf{w})\right\}^{1-t}$$

Training

- Depends on the task
- **For binary classification**: We use a logistic sigmoid outputs and a cross-entropy error function
- Output is $y = \sigma(a) \equiv \dfrac{1}{1 + \exp(-a)}$
  - The conditional distribution is given by a Bernoulli distribution

$$p(t|\mathbf{x}, \mathbf{w}) = y(\mathbf{x}, \mathbf{w})^t \{1 - y(\mathbf{x}, \mathbf{w})\}^{1-t}$$

  - Learn the parameters (maximizing -log likelihood as usual)

$$E(\boxed{\mathbf{w}}) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$$

Training

- Depends on the task
- **For multiclass classification**: We use a softmax output and a multiclass cross-entropy error function

Training

- Depends on the task
- **For multiclass classification**: We use a softmax output and a multiclass cross-entropy error function
- Output is $$y_k(\mathbf{x}, \mathbf{w}) = \frac{\exp(a_k(\mathbf{x}, \mathbf{w}))}{\sum_j \exp(a_j(\mathbf{x}, \mathbf{w}))}$$
  - The conditional distribution is given by

$$y_k(\mathbf{x}, \mathbf{w}) = p(t_k = 1|\mathbf{x})$$

Training

- Depends on the task
- **For multiclass classification**: We use a softmax output and a multiclass cross-entropy error function
- Output is $$y_k(\mathbf{x}, \mathbf{w}) = \frac{\exp(a_k(\mathbf{x}, \mathbf{w}))}{\sum_j \exp(a_j(\mathbf{x}, \mathbf{w}))}$$
  - The conditional distribution is given by

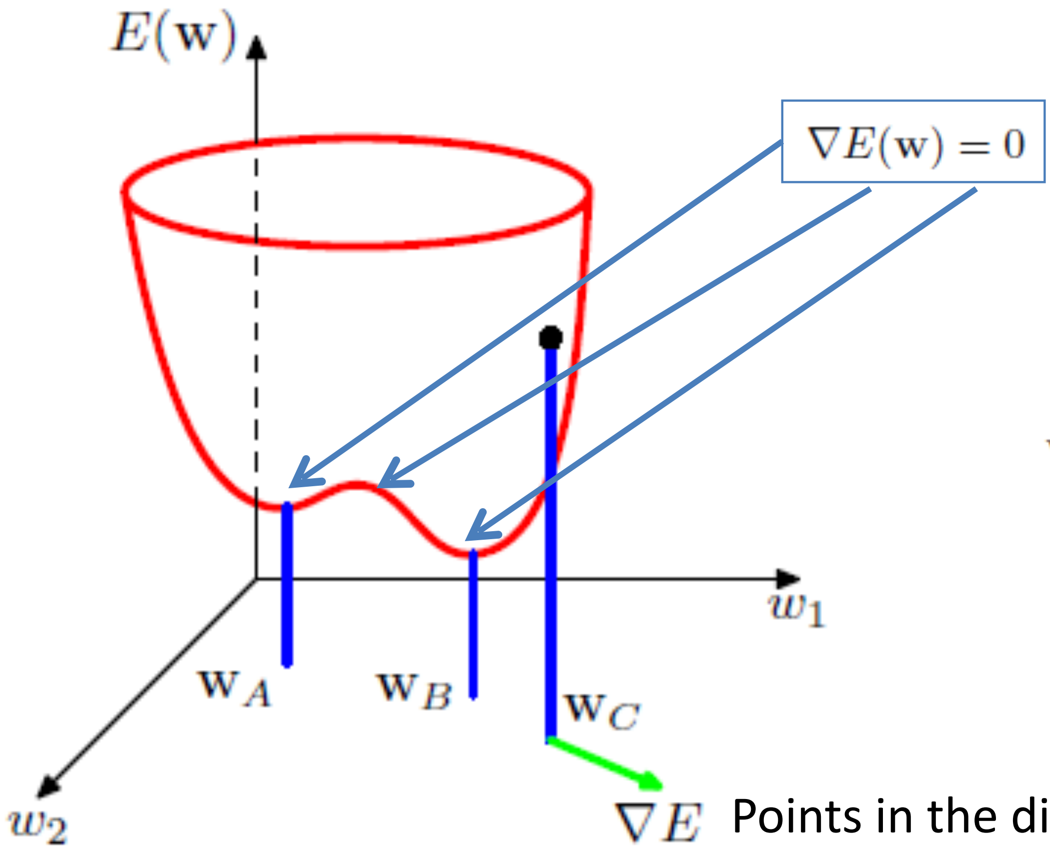
$$y_k(\mathbf{x}, \mathbf{w}) = p(t_k = 1|\mathbf{x})$$

  - Learn the parameters

$$E(\boxed{\mathbf{w}}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{kn} \ln y_k(\mathbf{x}_n, \mathbf{w})$$

Training

- Parameter optimization



$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau)}$$

Points in the direction
of greatest rate of increase
of the error function

Training

- Parameter optimization
- Using gradient information
  - More efficient, let's see: We wanted to evaluate

$$E(\mathbf{w}) \simeq E(\widehat{\mathbf{w}}) + (\mathbf{w} - \widehat{\mathbf{w}})^{\mathrm{T}}\mathbf{b} + \frac{1}{2}(\mathbf{w} - \widehat{\mathbf{w}})^{\mathrm{T}}\mathbf{H}(\mathbf{w} - \widehat{\mathbf{w}})$$

    - Between b and $\mathbf{H}$ we have $W(W+3)/2$ (W = dimensionality of w)
    - This means, that evaluate this function has complexity $O(W^3)$

Training

- Parameter optimization
- Using gradient information
  - More efficient, let's see: We wanted to evaluate

$$E(\mathbf{w}) \simeq E(\widehat{\mathbf{w}}) + (\mathbf{w} - \widehat{\mathbf{w}})^{\mathrm{T}}\mathbf{b} + \frac{1}{2}(\mathbf{w} - \widehat{\mathbf{w}})^{\mathrm{T}}\mathbf{H}(\mathbf{w} - \widehat{\mathbf{w}})$$

    - Between b and $\mathbf{H}$ we have $W(W+3)/2$ (W = dimensionality of w)
    - This means, that evaluate this function has complexity $O(W^3)$
  - One easy, more efficient way: *Gradient descent*

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)})$$

    - Requires the whole training set: *Batch method*
    - At each step the weight vector is moved in the direction of the greatest rate of decrease of the error function
    - In the end, it is not so efficient
      - Depends on the randomly chosen starting point

Neural networks

Feed-forward networks

Training

Backpropagation

Deep Learning

Training

- Parameter optimization
- Using gradient information: Error backpropagation
  - An efficient method to evaluate gradient information $O(W^2)$
- Two steps
  1. Evaluate the derivates of the error function with respect to the weights
     - Do this iteratively
     - For this we use gradient *backpropagation*: Error are propagated backwards through the network
  2. Derivatives are then used to compute the adjustments to be made to the weights
     - For this we can use *gradient descent*

Training

- Parameter optimization
- Using gradient information: Error backpropagation
  - Let's consider the following scenario
    - Output

$$y_k = \sum_i w_{ki} x_i$$

    - Error

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}) \qquad E_n = \frac{1}{2}\sum_k (y_{nk} - t_{nk})^2 \qquad \frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj}) x_{ni}$$

    - Activation function. z_i is the Activation of a unit, sending a connection to unit j

$$a_j = \sum_i \boxed{w_{ji}} z_i \qquad\qquad z_j = h(a_j)$$

We already have these from forward propagation

Training

- Parameter optimization
- Using gradient information: Error backpropagation
  - Let's focus on the derivative of the error

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj}) x_{ni} \qquad\qquad a_j = \sum_i w_{ji} z_i$$

Training

- Parameter optimization
- Using gradient information: Error backpropagation
  - Let's focus on the derivative of the error

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj})x_{ni} \qquad\qquad a_j = \sum_i w_{ji} z_i$$

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i$$

Training

- Parameter optimization
- Using gradient information: Error backpropagation
  - Let's focus on the derivative of the error

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj})x_{ni} \qquad\qquad a_j = \sum_i w_{ji} z_i$$

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i$$

  - We have to evaluate the errors (δ)
    - For the outputs, this is

$$\delta_k = y_k - t_k$$

Training

- Parameter optimization
- Using gradient information: Error backpropagation
  - Let's focus on the derivative of the error

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj})x_{ni} \qquad\qquad a_j = \sum_i w_{ji} z_i$$

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i$$

  - We have to evaluate the errors (δ)
    - For the outputs, this is

$$\delta_k = y_k - t_k$$

    - For the hidden units

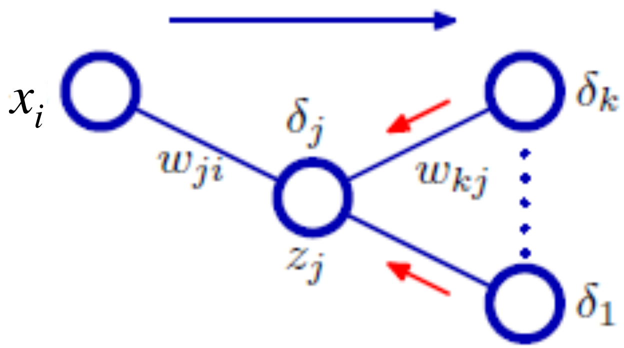
$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j}$$

Training

- Parameter optimization
- Using gradient information: Error backpropagation
  - Let's focus on the derivative of the error

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj})x_{ni} \qquad a_j = \sum_i w_{ji} z_i$$

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i$$



  - We have to evaluate the errors (δ)
    - For the outputs, this is

$$\boxed{\delta_k} = y_k - t_k$$

    - For the hidden units

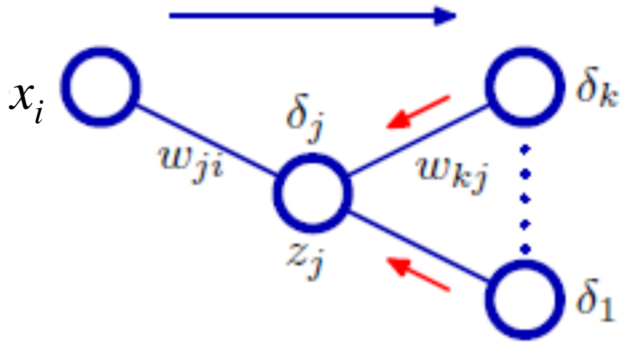
$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \boxed{\frac{\partial E_n}{\partial a_k}}\frac{\partial a_k}{\partial a_j} = h'(a_j)\sum_k w_{kj}\boxed{\delta_k}$$

Training

- Parameter optimization
- Using gradient information: Error backpropagation
  - Let's focus on the derivative of the error

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj})x_{ni} \qquad a_j = \sum_i w_{ji} z_i$$

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i$$



  - We have to evaluate the errors (δ)
    - For the outputs, this is

$$\delta_k = y_k - t_k$$

    - For the hidden units

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \boxed{\frac{\partial a_k}{\partial a_j}} = h'(a_j) \sum_k \boxed{w_{kj}} \delta_k$$

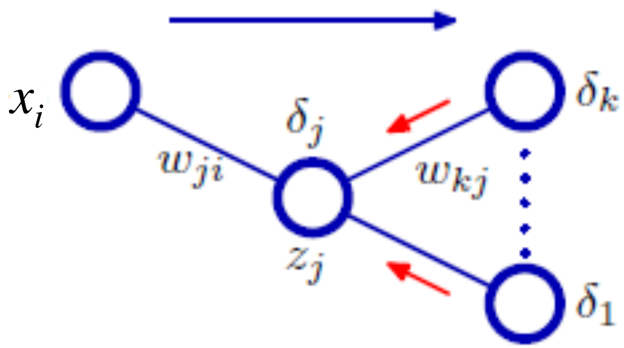
Training

- Parameter optimization
- Using gradient information: Error backpropagation
  - Let's focus on the derivative of the error

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj})x_{ni}$$

$$a_j = \sum_i w_{ji} z_i$$

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i$$



  - We have to evaluate the errors (δ)
    - For the outputs, this is

$$\delta_k = y_k - t_k$$

    - For the hidden units

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\boxed{\frac{\partial a_k}{\partial a_j}} = \boxed{h'(a_j)}\sum_k \boxed{w_{kj}}\delta_k$$

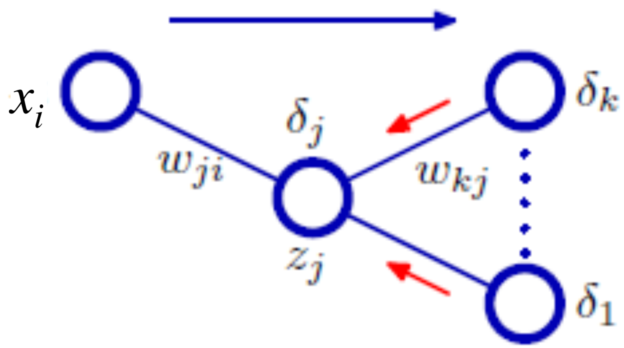
$$z_j = h(a_j)$$

Training

- Parameter optimization
- Using gradient information: Error backpropagation
  - Let's focus on the derivative of the error

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj})x_{ni} \qquad a_j = \sum_i w_{ji} z_i$$

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i$$



  - We have to evaluate the errors (δ)
    - For the outputs, this is

$$\delta_k = y_k - t_k$$

    - For the hidden units

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} = \boxed{h'(a_j) \sum_k w_{kj} \delta_k}$$

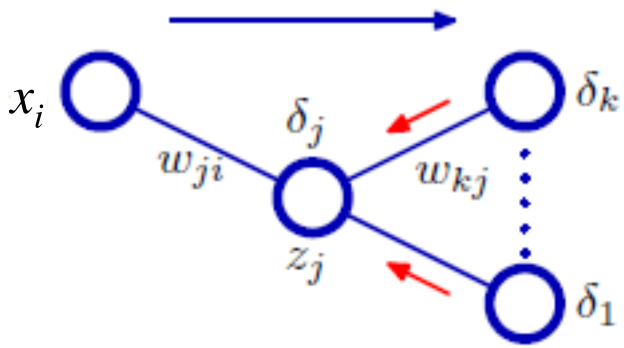
    - This is the backpropagation formula

Training

- Parameter optimization
- Using gradient information: Error backpropagation
  - Let's focus on the derivative of the error

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj})x_{ni} \qquad a_j = \sum_i w_{ji} z_i$$

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i$$



  - We have to evaluate the errors (δ)
    - For the outputs, this is

$$\delta_k = y_k - t_k$$

    - For the hidden units

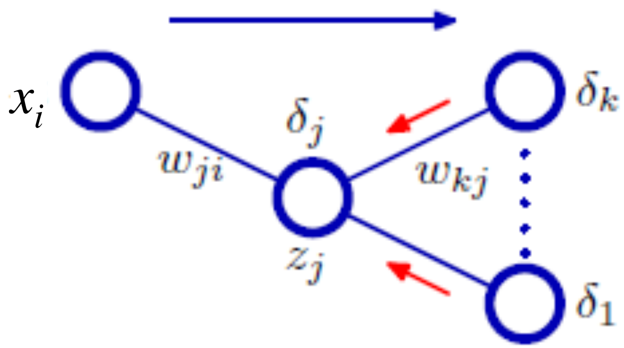
$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} = \boxed{h'(a_j)\sum_k w_{kj}\delta_k}$$

    - This is the backpropagation formula
    - Remember the forward propagation was $z_k = h\left(\sum_j w_{kj} z_j\right)$

Training

- Parameter optimization
- Using gradient information: Error backpropagation
  - An example
    - Output units we have $y_k = a_k$
    - For hidden units

$$h(a) \equiv \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$

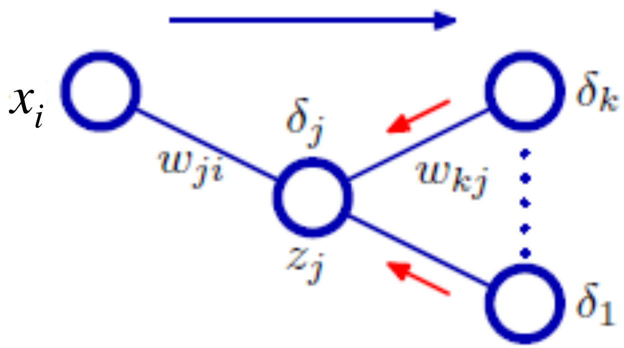


Training

- Parameter optimization
- Using gradient information: Error backpropagation
  - An example
    - Output units we have $y_k = a_k$
    - For hidden units

$$h(a) \equiv \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$

– Its derivative is $h'(a) = 1 - h(a)^2$



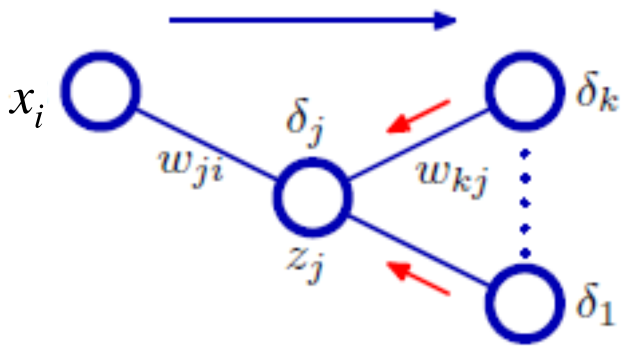
Training

- Parameter optimization
- Using gradient information: Error backpropagation
  - An example
    - Output units we have $y_k = a_k$
    - For hidden units

$$h(a) \equiv \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$

      - Its derivative is $h'(a) = 1 - h(a)^2$
    - Sum-of-squares error

$$E_n = \frac{1}{2}\sum_{k=1}^{K}(y_k - t_k)^2$$



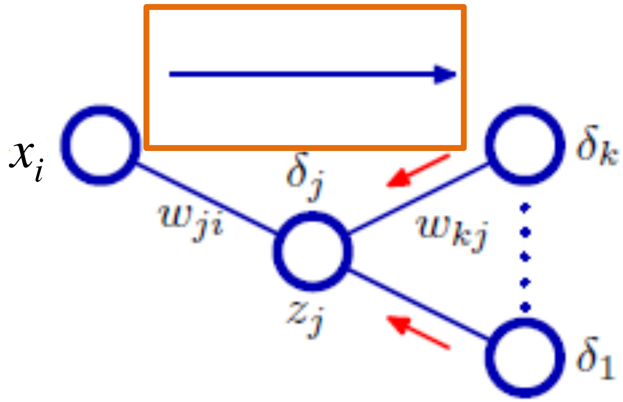
Training

- Parameter optimization
- Using gradient information: Error backpropagation
  - An example
    - Output units we have $y_k = a_k$
    - For hidden units

$$h(a) \equiv \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$

      - Its derivative is $h'(a) = 1 - h(a)^2$
    - Sum-of-squares error

$$E_n = \frac{1}{2}\sum_{k=1}^{K}(y_k - t_k)^2$$



1. Forward propagation

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i \qquad z_j = \tanh(a_j)$$

$$y_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j$$

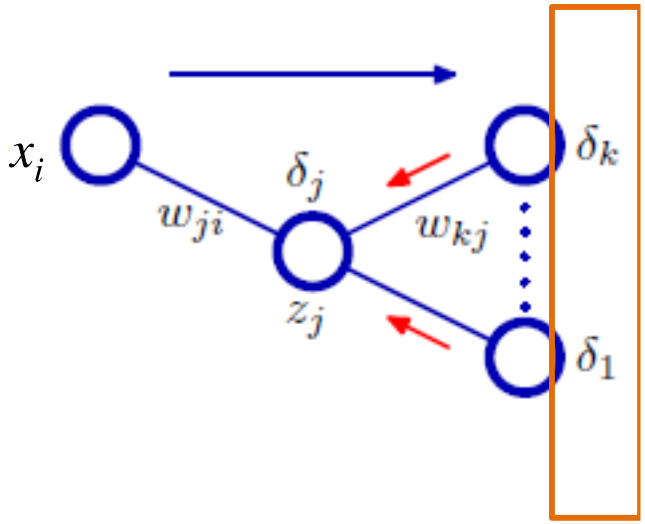
Training

- Parameter optimization
- Using gradient information: Error backpropagation
  - An example
    - Output units we have $y_k = a_k$
    - For hidden units

$$h(a) \equiv \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$

      - Its derivative is $h'(a) = 1 - h(a)^2$
    - Sum-of-squares error

$$E_n = \frac{1}{2}\sum_{k=1}^{K}(y_k - t_k)^2$$



1. Forward propagation

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i \qquad z_j = \tanh(a_j)$$

$$y_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j$$

2. Compute errors

$$\delta_k = y_k - t_k$$

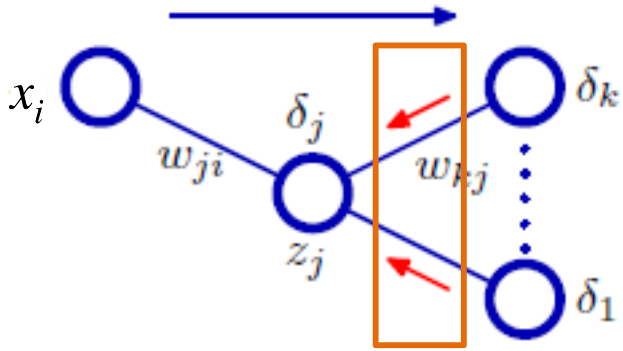
Training

- Parameter optimization
- Using gradient information: Error backpropagation
  - An example
    - Output units we have $y_k = a_k$
    - For hidden units

$$h(a) \equiv \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$

      - Its derivative is $h'(a) = 1 - h(a)^2$
    - Sum-of-squares error

$$E_n = \frac{1}{2}\sum_{k=1}^{K}(y_k - t_k)^2$$



1. Forward propagation

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i \qquad z_j = \tanh(a_j)$$

$$y_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j$$

2. Compute errors

$$\delta_k = y_k - t_k$$

3. Backpropagate

$$\delta_j = (1 - z_j^2)\sum_{k=1}^{K} w_{kj}\delta_k$$

$$= h'(a_j)\sum_{k} w_{kj}\delta_k$$

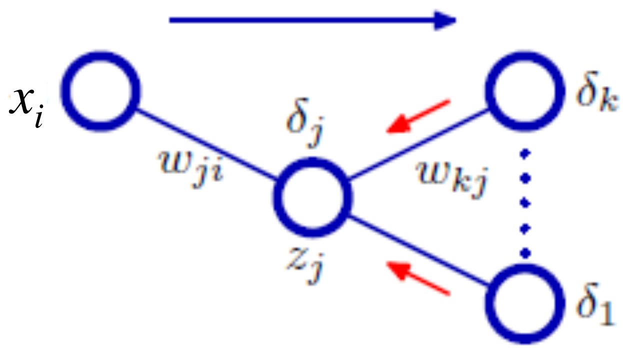
Training

- Parameter optimization
- Using gradient information: Error backpropagation
  - An example
    - Output units we have $y_k = a_k$
    - For hidden units

$$h(a) \equiv \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$

      - Its derivative is $h'(a) = 1 - h(a)^2$
    - Sum-of-squares error

$$E_n = \frac{1}{2}\sum_{k=1}^{K}(y_k - t_k)^2$$



    - Finally, the derivatives with respect to the first and second layer weights are

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i, \qquad \frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j$$

Training

- Parameter optimization
- One easy, efficient way: *Gradient descent*
- Two steps
  1. Evaluate the derivates of the error function with respect to the weights
     - Do this iteratively
     - For this we use gradient *backpropagation*: Error are propagated backwards through the network

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i, \qquad \frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j$$

  2. Derivatives are then used to compute the adjustments to be made to the weights
     - For this we can use *gradient descent*

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)})$$

Training

- Parameter optimization
- One easy, efficient way: *Gradient descent*
- Two steps
  1. Evaluate the derivates of the error function with respect to the weights
     - Do this iteratively
     - For this we use gradient *backpropagation*: Error are propagated backwards through the network

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i, \qquad \frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j$$

  2. Derivatives are then used to compute the adjustments to be made to the weights
     - For this we can use *gradient descent*

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)})$$

Training

- https://www.youtube.com/watch?v=Ilg3gGewQ5U
- One easy, efficient way: *Gradient descent*
- Two steps
  1. Evaluate the derivates of the error function with respect to the weights
     - Do this iteratively
     - For this we use gradient *backpropagation*: Error are propagated backwards through the network

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i, \qquad \frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j$$

  2. Derivatives are then used to compute the adjustments to be made to the weights
     - For this we can use *gradient descent*

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)})$$

Neural networks

Feed-forward networks

Training

Backpropagation

Deep Learning

Deep Learning

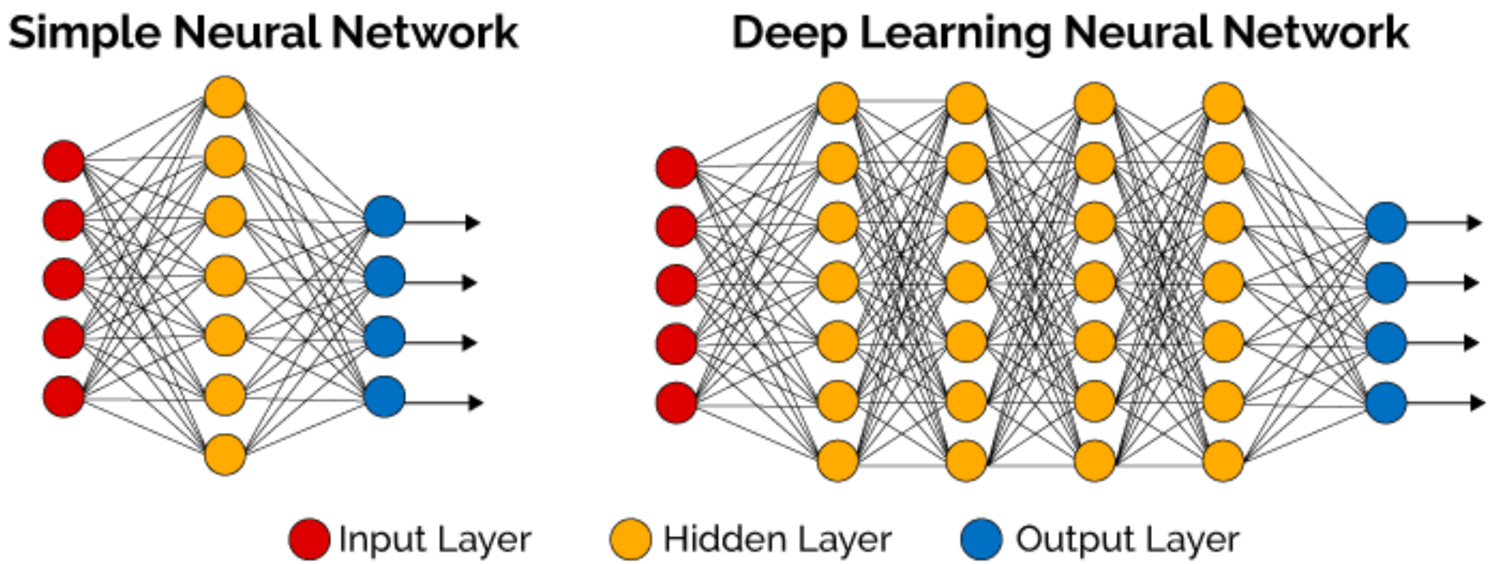
- Allow computers to learn from experience
- Understand the world in terms of a hierarchy of concepts
  - Each concept is defined through its relation to simpler concepts
  - The hierarchy of concepts enables the computer to learn concepts building them out of those simpler ones

Deep Learning

- Allow computers to learn from experience
- Understand the world in terms of a hierarchy of concepts
  - Each concept is defined through its relation to simpler concepts
  - The hierarchy of concepts enables the computer to learn concepts building them out of those simpler ones
- If we draw this as a graph, the graph is deep

Deep Learning

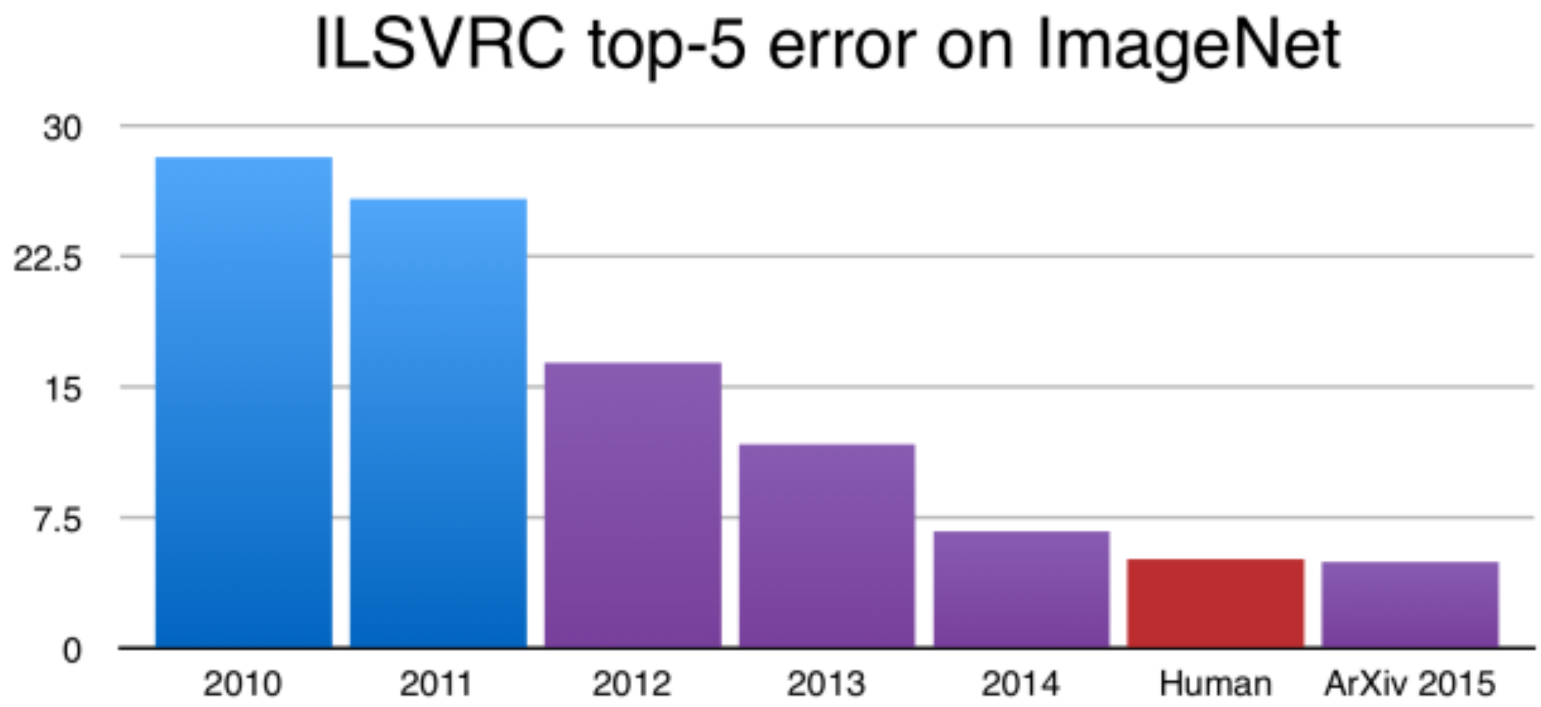
- If we draw those characteristics into a graph, the graph is deep



Convolutional Neural Networks (CNN)

AlexNet (2012)

- Bigger datasets: more training
- More processing power: GPUs (Graphical Processing Units)
- In combination with deep neural networks have become unbeatable



Convolutional Neural Networks (CNN)

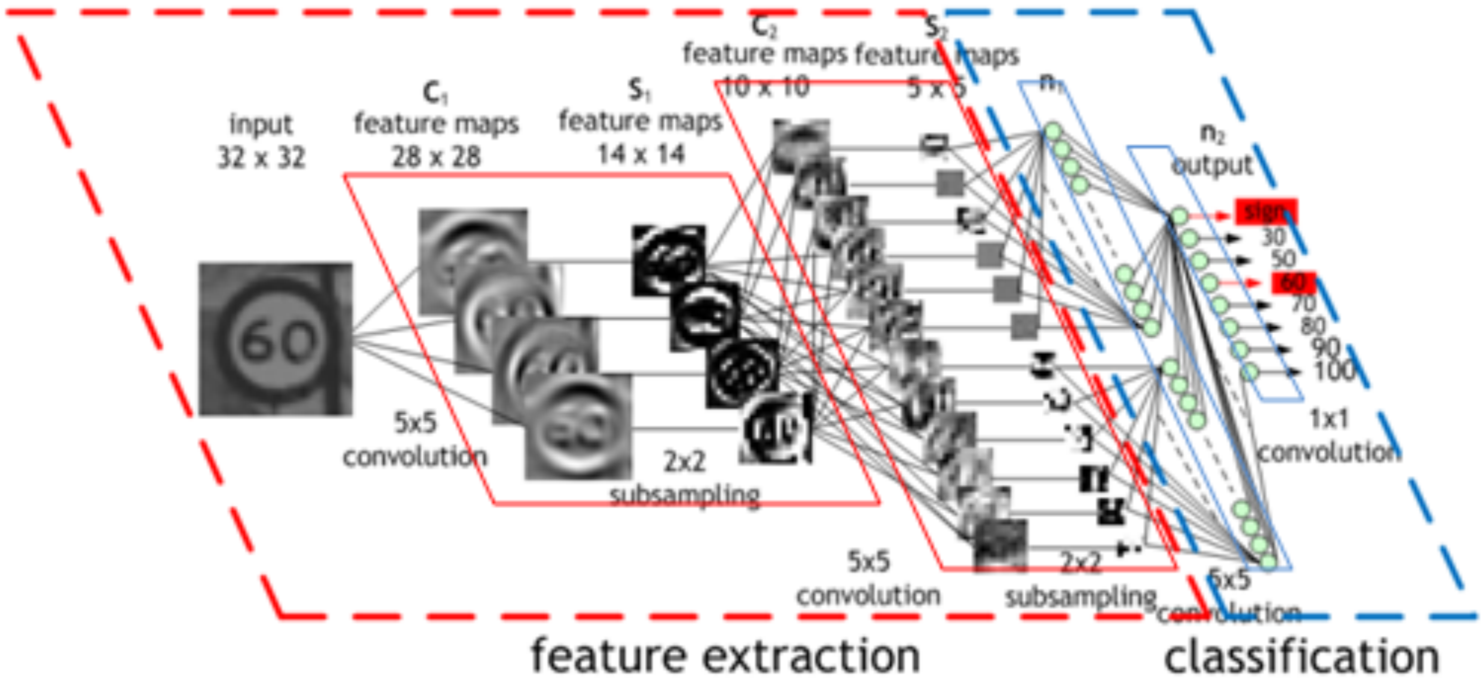
A CNN is a neural network but …

- Assumes the inputs are images, this has the following effects
  - Weight replication: The number of parameters is reduced
  - Thus, they are more efficient

Convolutional Neural Networks (CNN)

A CNN is a neural network but …

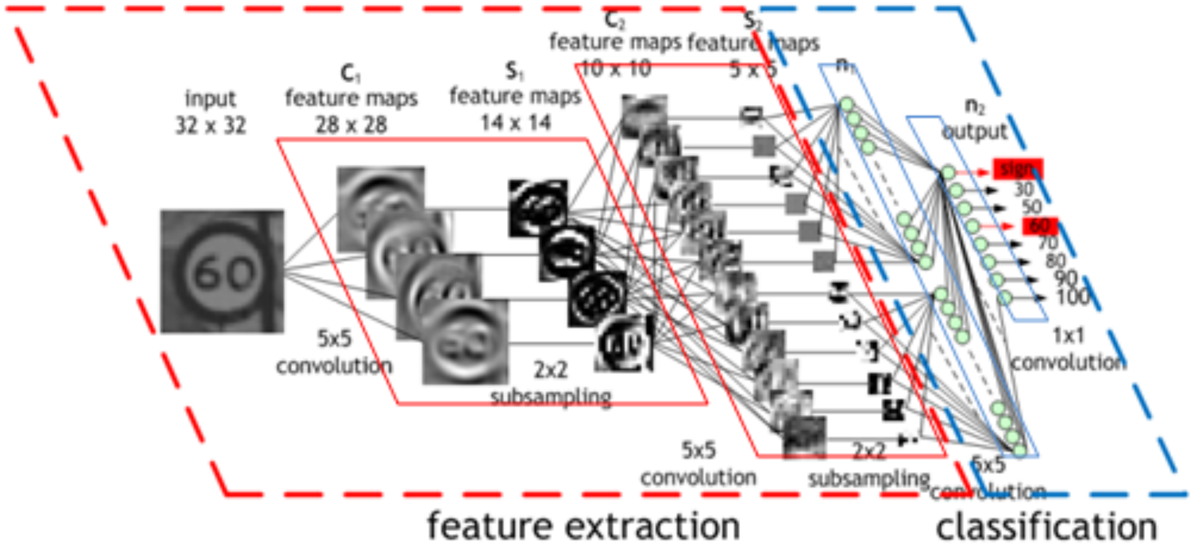
- Assumes the inputs are images, this has the following effects
  - Weight replication: The number of parameters is reduced
  - Thus, they are more efficient



Convolutional Neural Networks (CNN)

Types of layers

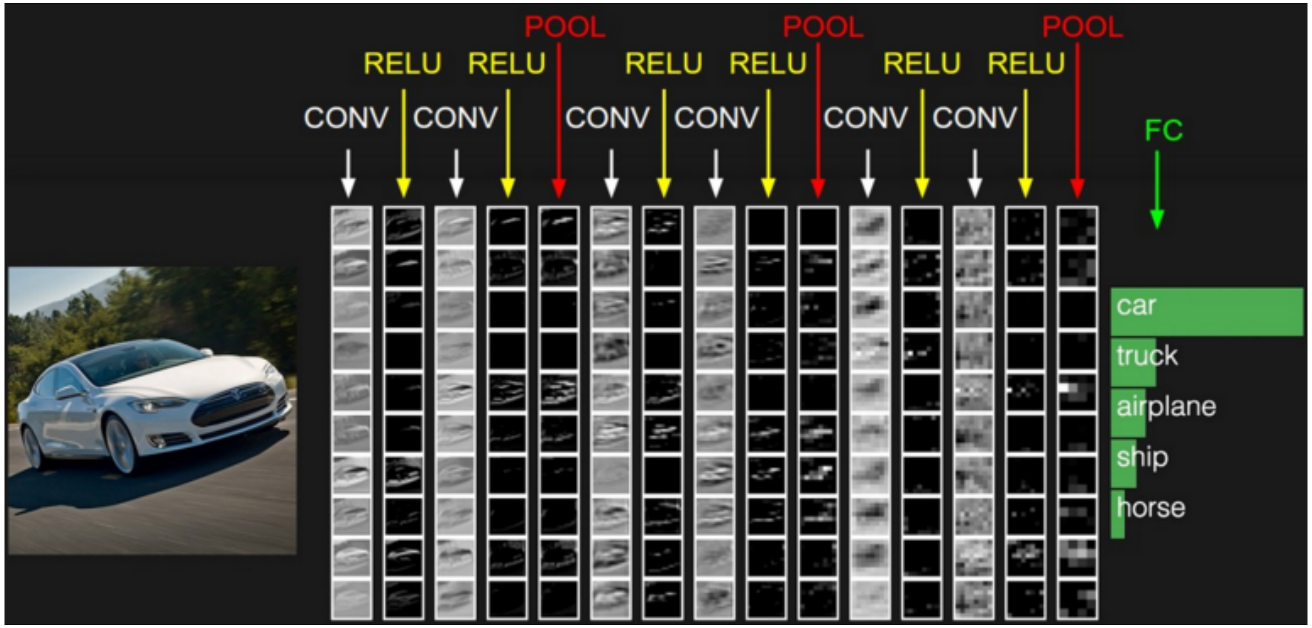
- Convolutional layer
- Nonlinearity
- Pooling layer
- Normalization layers
- Fully Connected layer
- Loss layer



Convolutional Neural Networks (CNN)

Types of layers

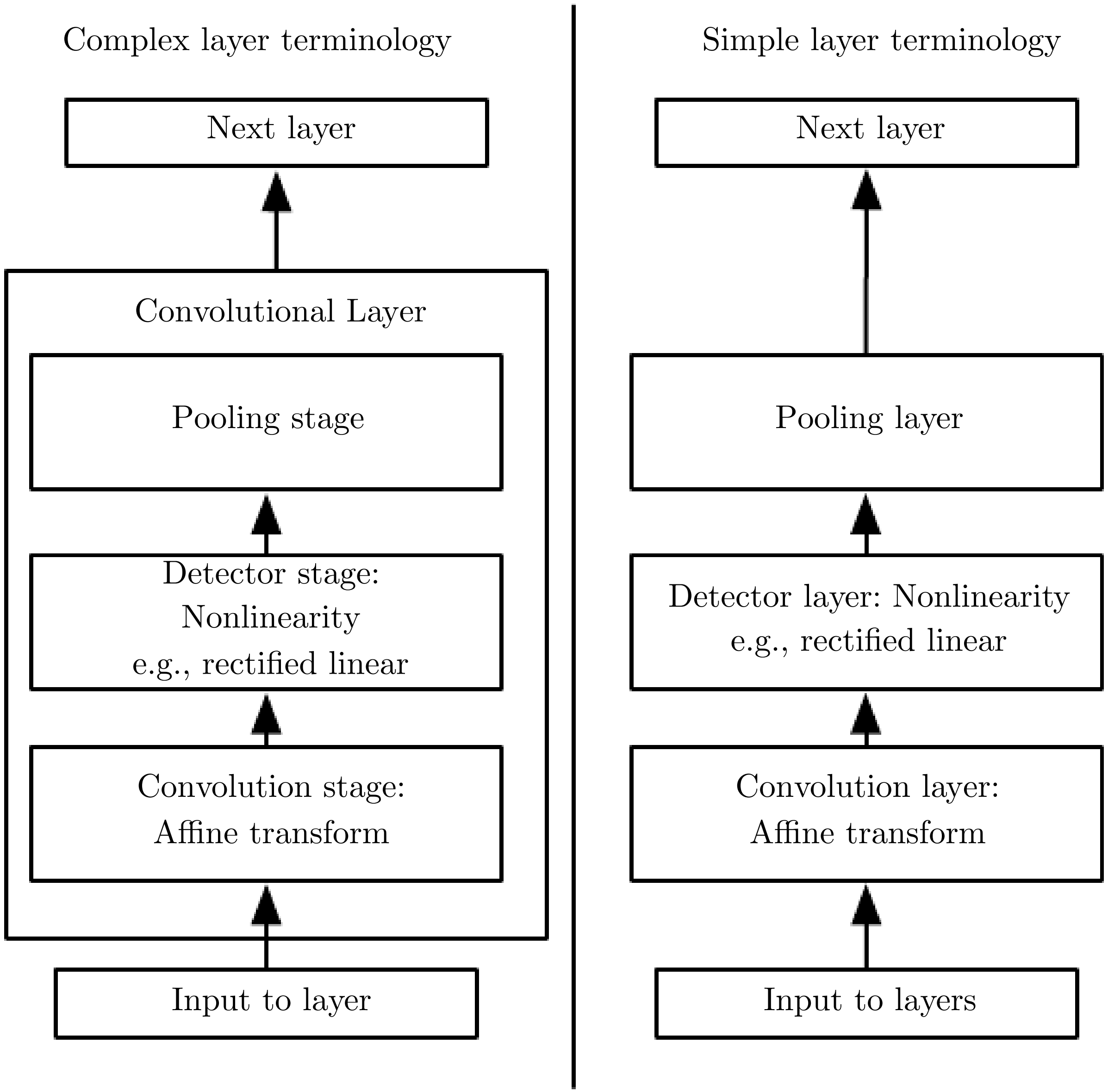
- Convolutional layer
- Nonlinearity
- Pooling layer
- Normalization layers
- Fully Connected layer
- Loss layer



Convolutional Neural Networks (CNN)

Types of layers

- Convolutional layer
- Nonlinearity
- Pooling layer
- Normalization layers
- Fully Connected layer
- Loss layer



Convolutional Neural Networks (CNN)

Types of layers

- Convolutional layer
- Nonlinearity
- Pooling layer
- Normalization layers
- Fully Connected layer
- Loss layer

Linear filtering

- Neighborhood operators
  - The output pixel's value is determined as a weighted sum of input pixel values

45	60	98	127	132	133	137	133
46	65	98	123	126	128	131	133
47	65	96	115	119	123	135	137
47	63	91	107	113	122	138	134
50	59	80	97	110	123	133	134
49	53	68	83	97	113	128	133
50	50	58	70	84	102	116	126
50	50	52	58	69	86	101	120

$f(x,y)$

*

0.1	0.1	0.1
0.1	0.2	0.1
0.1	0.1	0.1

$h(x,y)$

=

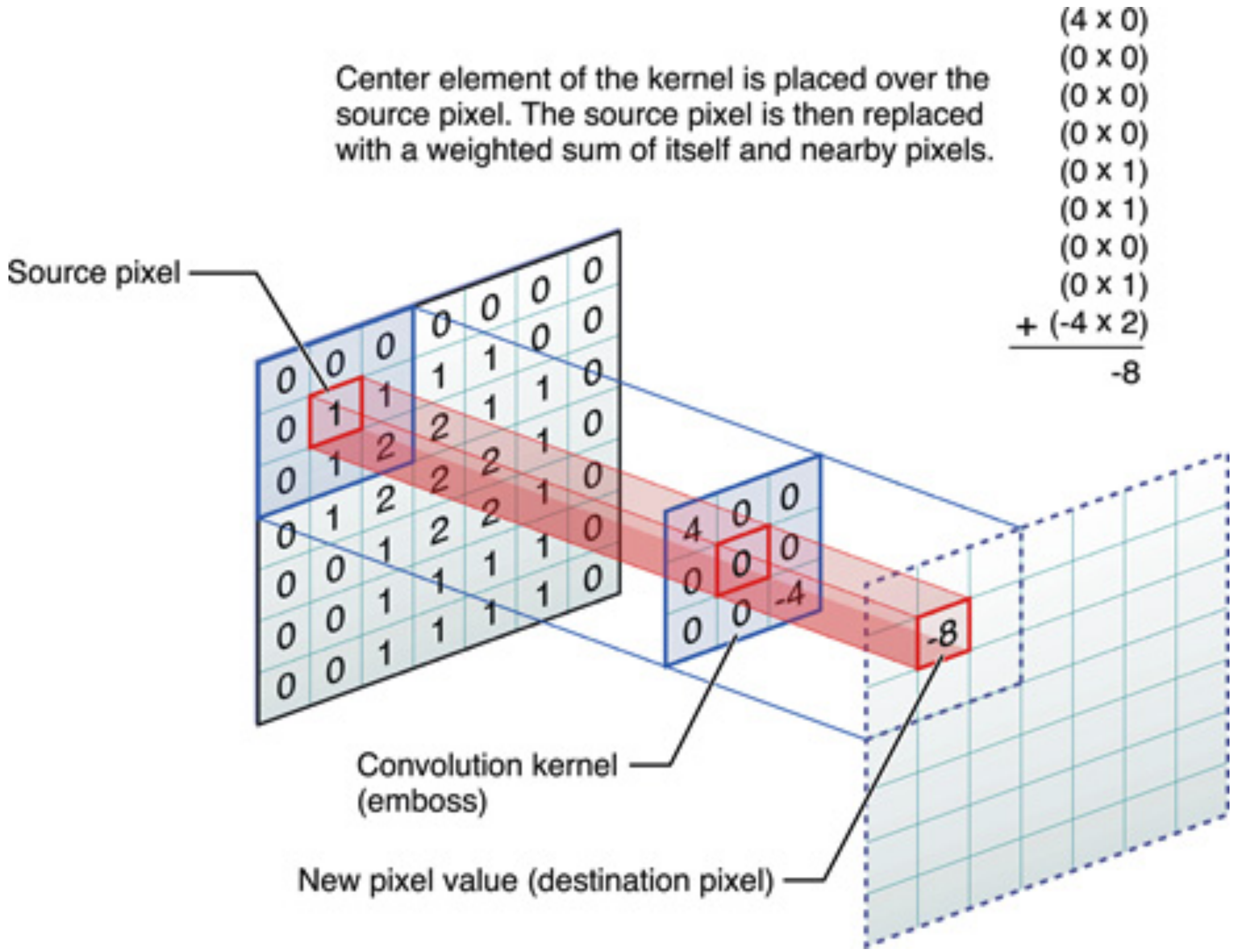
69	95	116	125	129	132
68	92	110	120	126	132
66	86	104	114	124	132
62	78	94	108	120	129
57	69	83	98	112	124
53	60	71	85	100	114

$g(x,y)$

Convolutional Neural Networks (CNN)

Types of layers

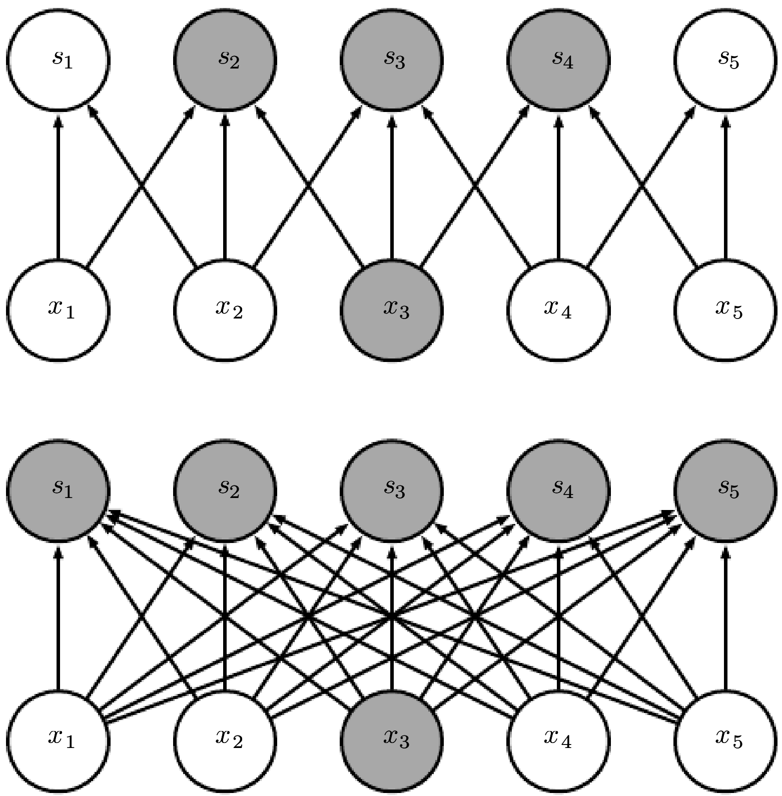
- Convolutional layer
  - Sparse interactions
  - Parameter sharing
  - Equivariant representations



Convolutional Neural Networks (CNN)

Types of layers

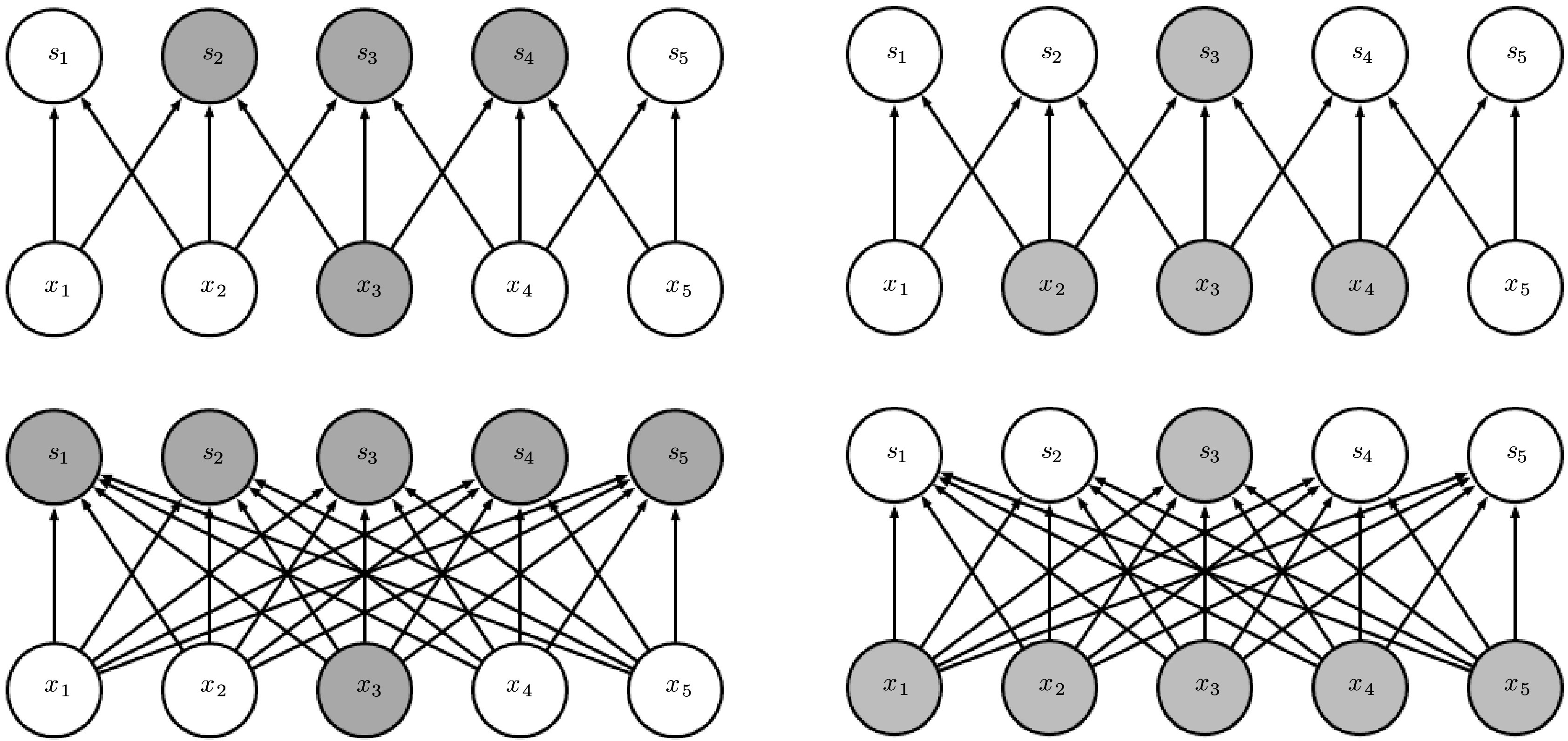
- Convolutional layer
  - Sparse interactions
    - Fewer parameters. $O(k \times n)$ instead of $O(m \times n)$, $k < m$



Convolutional Neural Networks (CNN)

Types of layers

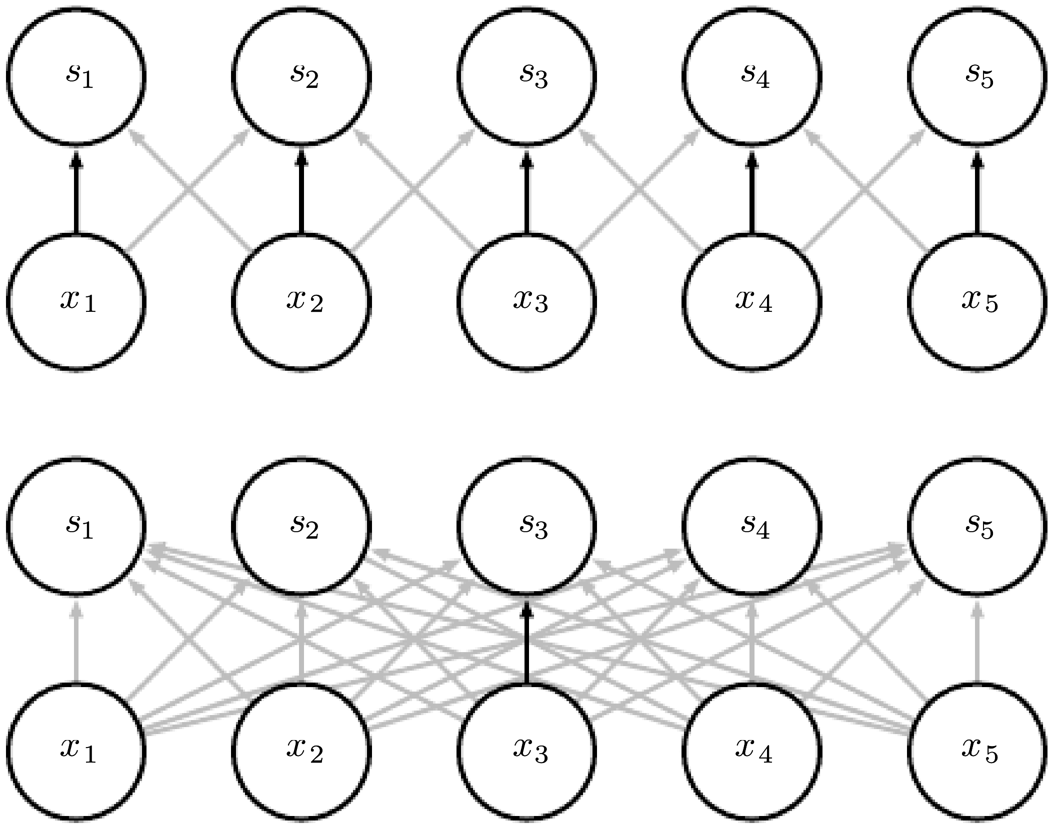
- Convolutional layer
  - Sparse interactions
    - Fewer parameters. $O(k \times n)$ instead of $O(m \times n)$, $k < m$



Convolutional Neural Networks (CNN)

Types of layers

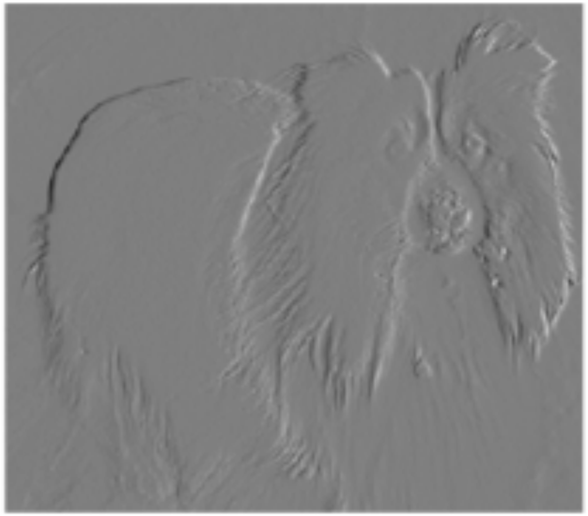
- Convolutional layer
  - Sparse interactions
  - Parameter sharing (Tied weights)



Convolutional Neural Networks (CNN)

Types of layers

- Convolutional layer
  - Sparse interactions
  - Parameter sharing (Tied weights)
    - Rather than learning a separate set of parameters for every location, we learn only one set

Convolutional Neural Networks (CNN)

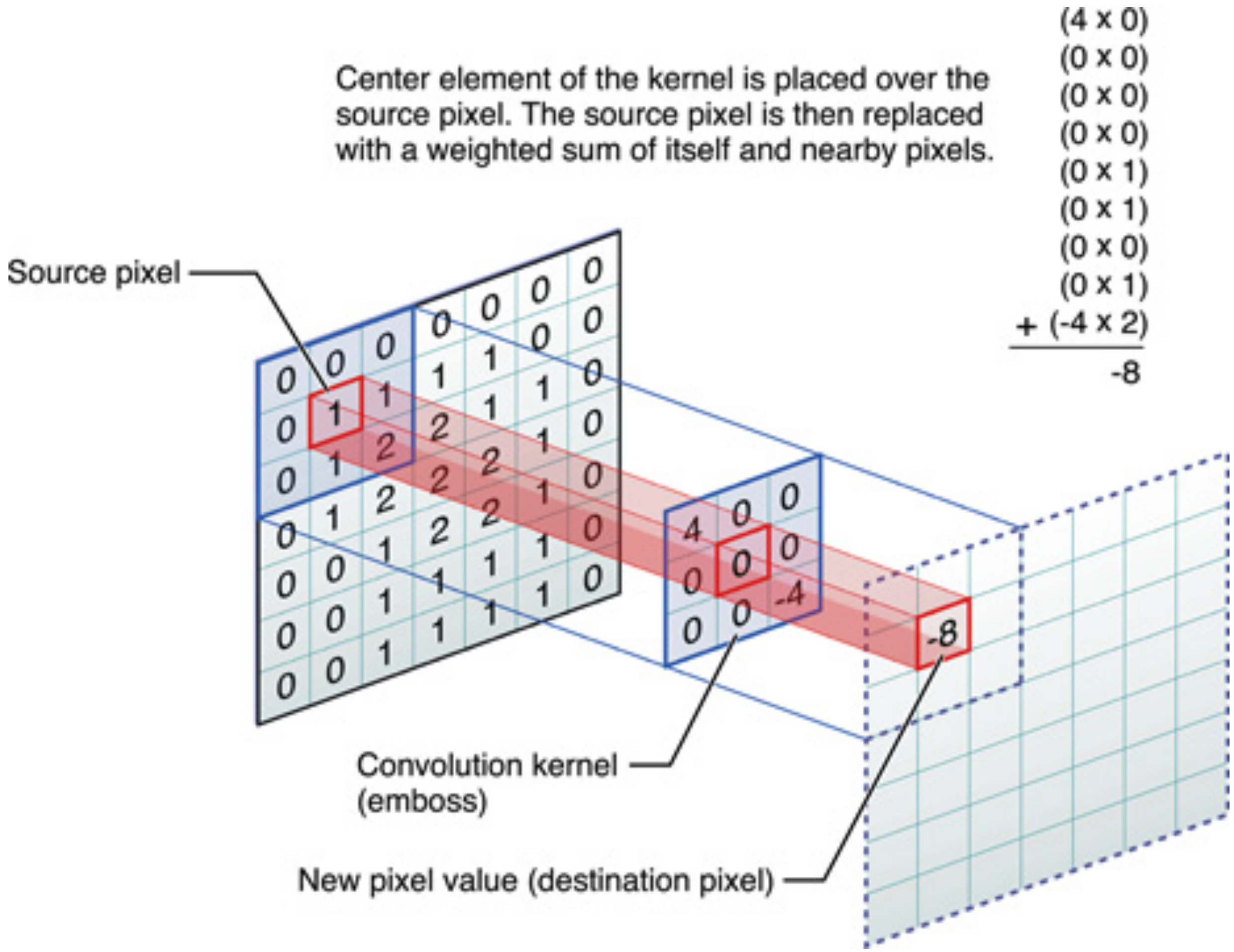
Types of layers

- Convolutional layer
  - Sparse interactions
  - Parameter sharing
  - Equivariance to translation

Convolutional Neural Networks (CNN)

Types of layers

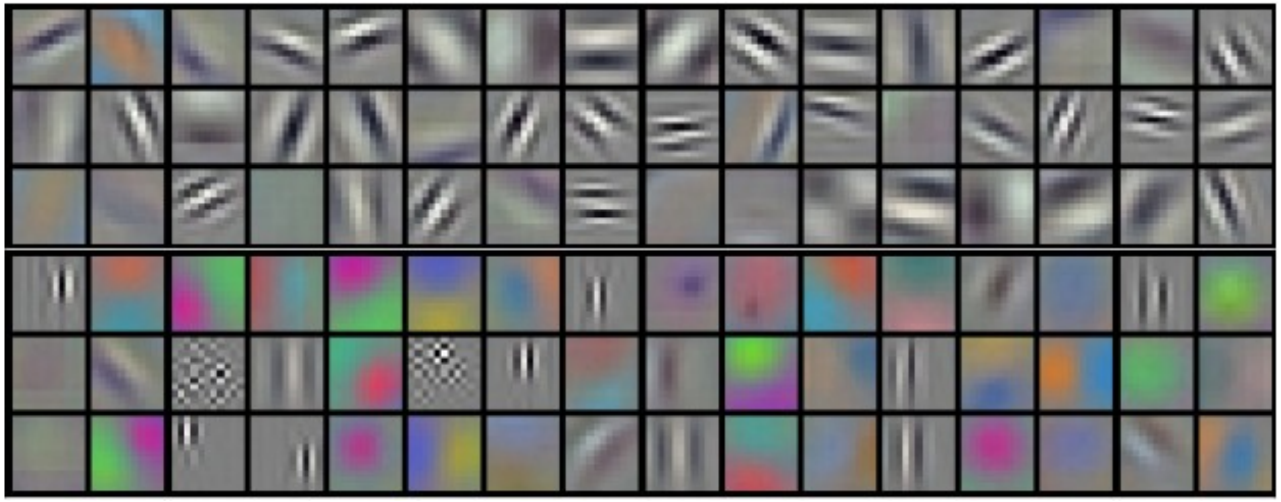
- Convolutional layer
  - Filters are learned
  - Hyperparameters
    - Size of filter
    - Depth (=number)
    - Stride
    - Padding



Convolutional Neural Networks (CNN)

Types of layers

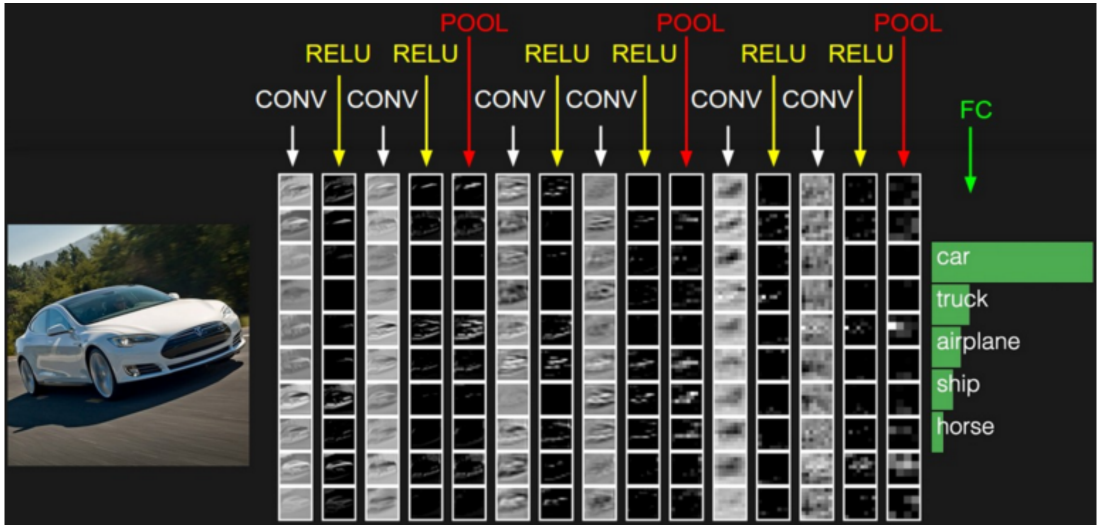
- Convolutional layer
  - Filters are learned
  - Hyperparameters
    - Size of filter
    - Depth (=number)
    - Stride
    - Padding



Convolutional Neural Networks (CNN)

Types of layers

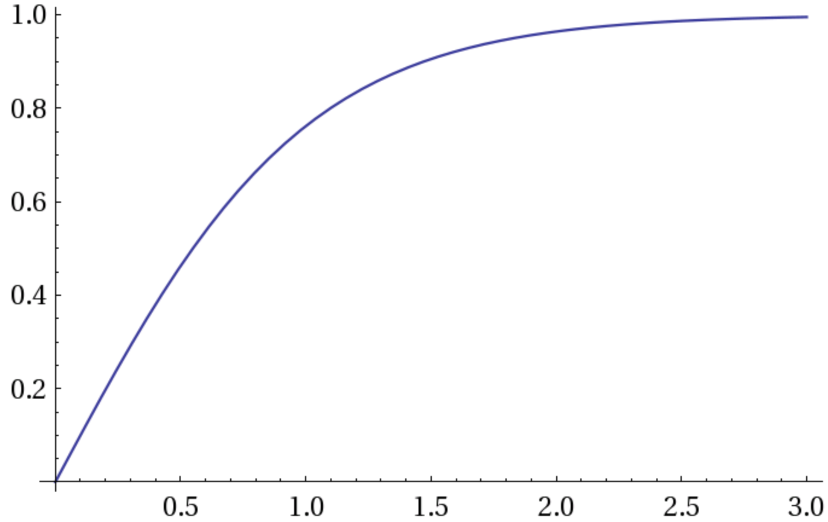
- Convolutional layer
- Nonlinearity
  - Hyperbolic tangent $\tanh(a) = \dfrac{e^a - e^{-a}}{e^a + e^{-a}}$
  - Sigmoid function $\sigma(a) = \dfrac{1}{1 + \exp(-a)}$
  - ReLU (Rectified Linear Unit)
    $y = \max(0,x)$
  - Others.



Convolutional Neural Networks (CNN)

Types of layers

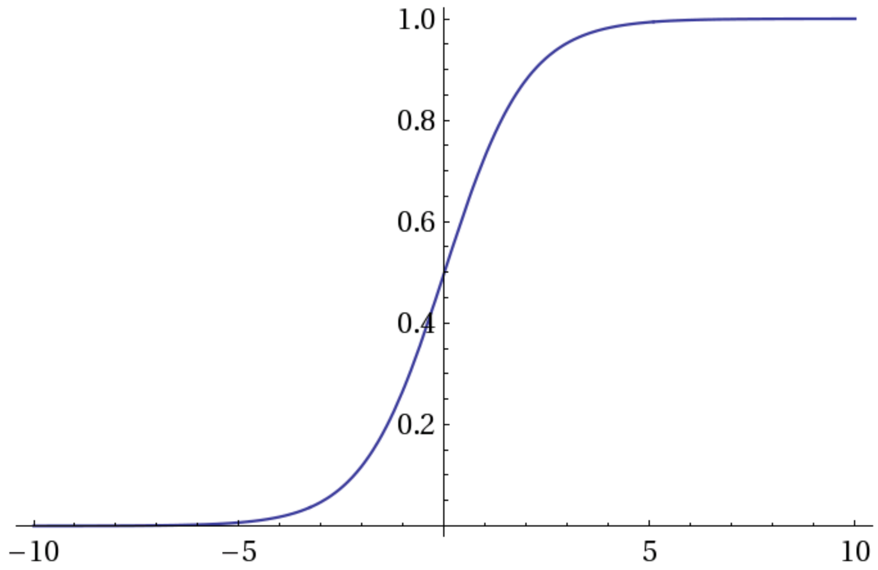
- Convolutional layer
- Nonlinearity
  - Hyperbolic tangent $\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$
  - Sigmoid function $\sigma(a) = \frac{1}{1 + \exp(-a)}$
  - ReLU (Rectified Linear Unit)
    $y = \max(0, x)$
  - Others.



Convolutional Neural Networks (CNN)

Types of layers

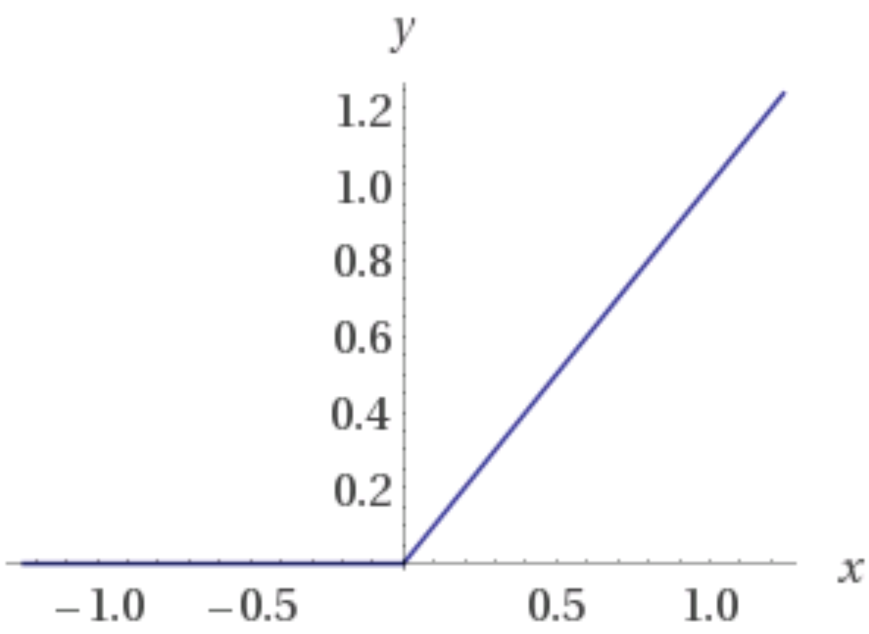
- Convolutional layer
- Nonlinearity
  - Hyperbolic tangent $\tanh(a) = \dfrac{e^a - e^{-a}}{e^a + e^{-a}}$
  - Sigmoid function $\sigma(a) = \dfrac{1}{1 + \exp(-a)}$
  - ReLU (Rectified Linear Unit)
    $y = \max(0, x)$
  - Others.



Convolutional Neural Networks (CNN)

Types of layers

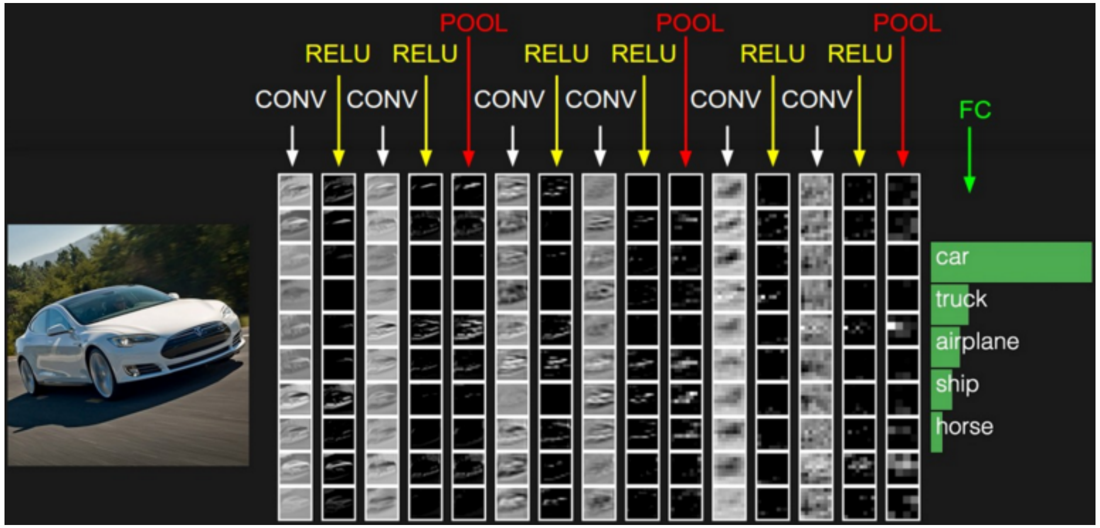
- Convolutional layer
- Nonlinearity
  - Hyperbolic tangent $\tanh(a) = \dfrac{e^a - e^{-a}}{e^a + e^{-a}}$
  - Sigmoid function $\sigma(a) = \dfrac{1}{1 + \exp(-a)}$
  - ReLU (Rectified Linear Unit)
    $y = \max(0, x)$
  - Others.



Convolutional Neural Networks (CNN)

Types of layers

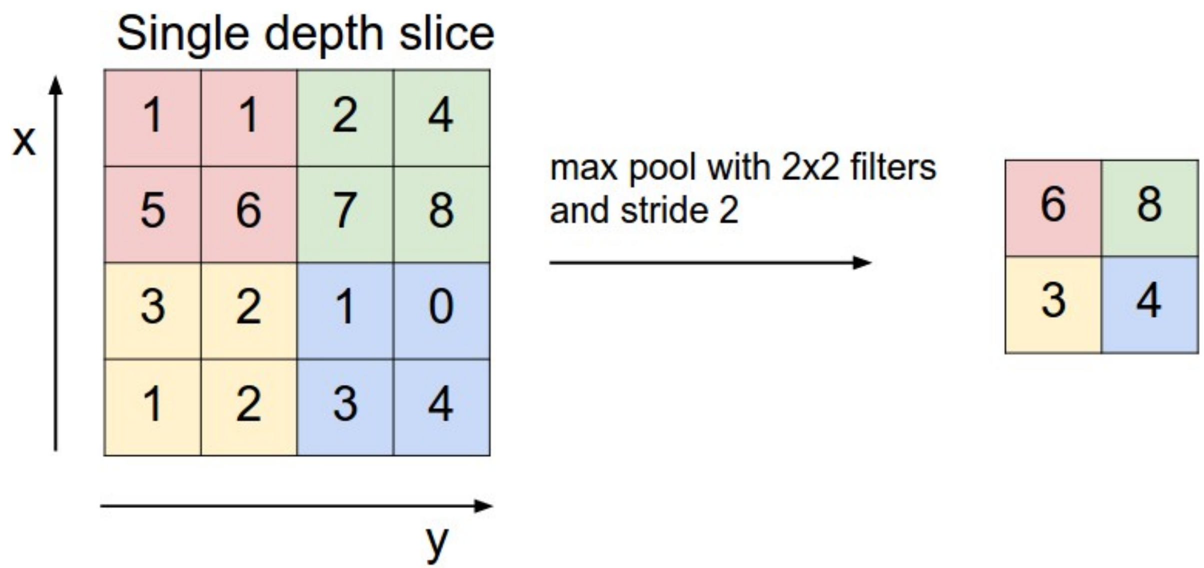
- Convolutional layer
- Nonlinearity
- Pooling layer
  - Reduce dimensionality
  - Position invariance
  - Max vs average pooling
    - Also L^2-norm
    - Fractional max-pooling



Convolutional Neural Networks (CNN)

Types of layers

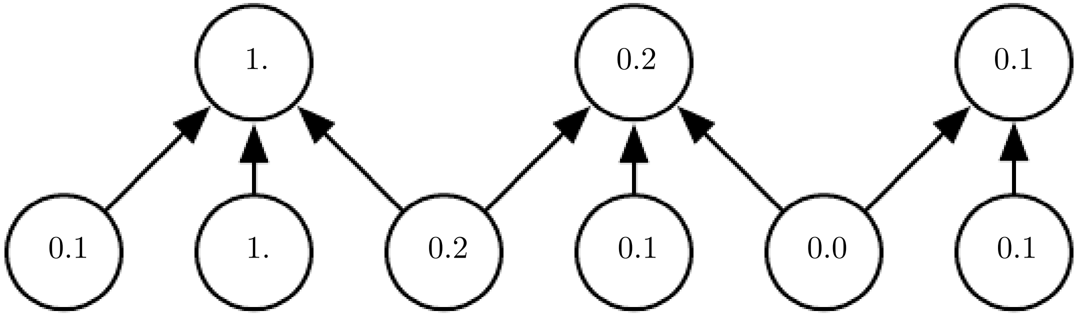
- Convolutional layer
- Nonlinearity
- Pooling layer
  - Reduce dimensionality
  - Position invariance
  - Max vs average pooling
    - Also L^2-norm
    - Fractional max-pooling



Convolutional Neural Networks (CNN)

Types of layers

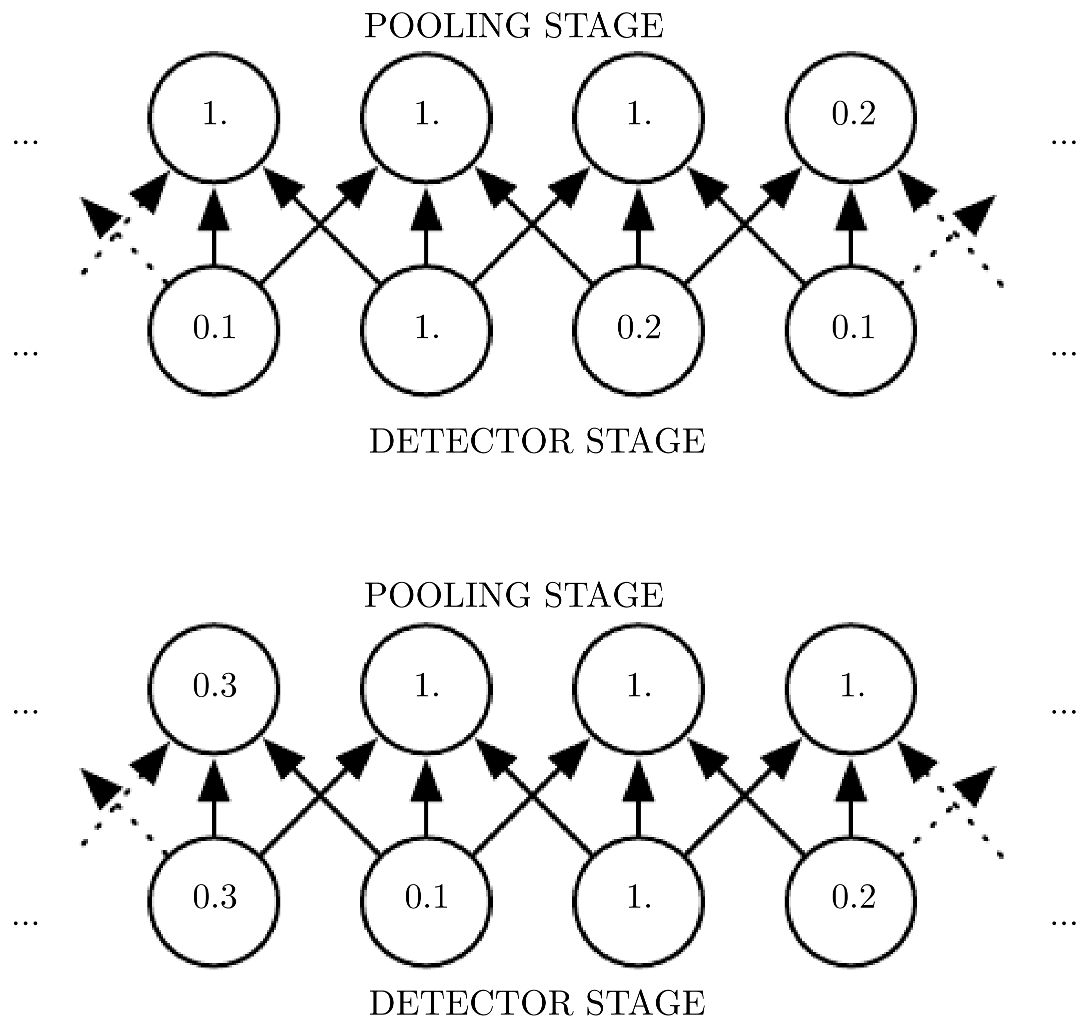
- Convolutional layer
- Nonlinearity
- Pooling layer
  - Reduce dimensionality
  - Position invariance
  - Max vs average pooling
    - Also L^2-norm
    - Fractional max-pooling



Convolutional Neural Networks (CNN)

Types of layers

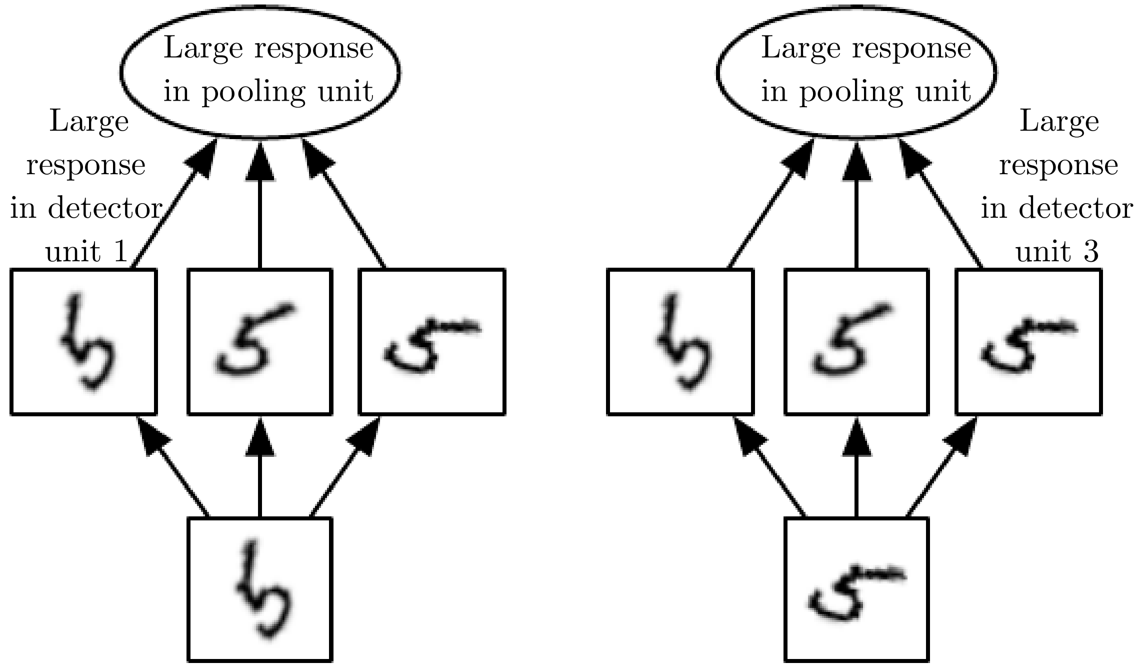
- Convolutional layer
- Nonlinearity
- Pooling layer
  - Reduce dimensionality
  - Position invariance
  - Max vs average pooling
    - Also L^2-norm
    - Fractional max-pooling



Convolutional Neural Networks (CNN)

Types of layers

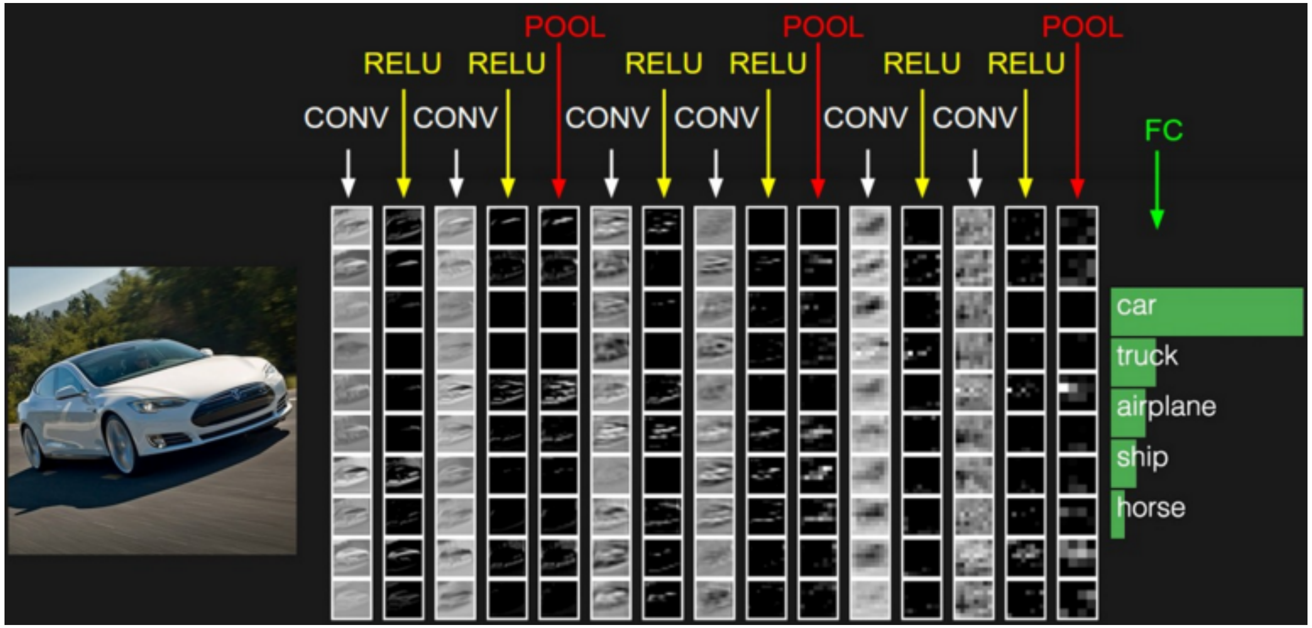
- Convolutional layer
- Nonlinearity
- Pooling layer
  - Reduce dimensionality
  - Position invariance
  - Max vs average pooling
    - Also L^2-norm
    - Fractional max-pooling



Convolutional Neural Networks (CNN)

Types of layers

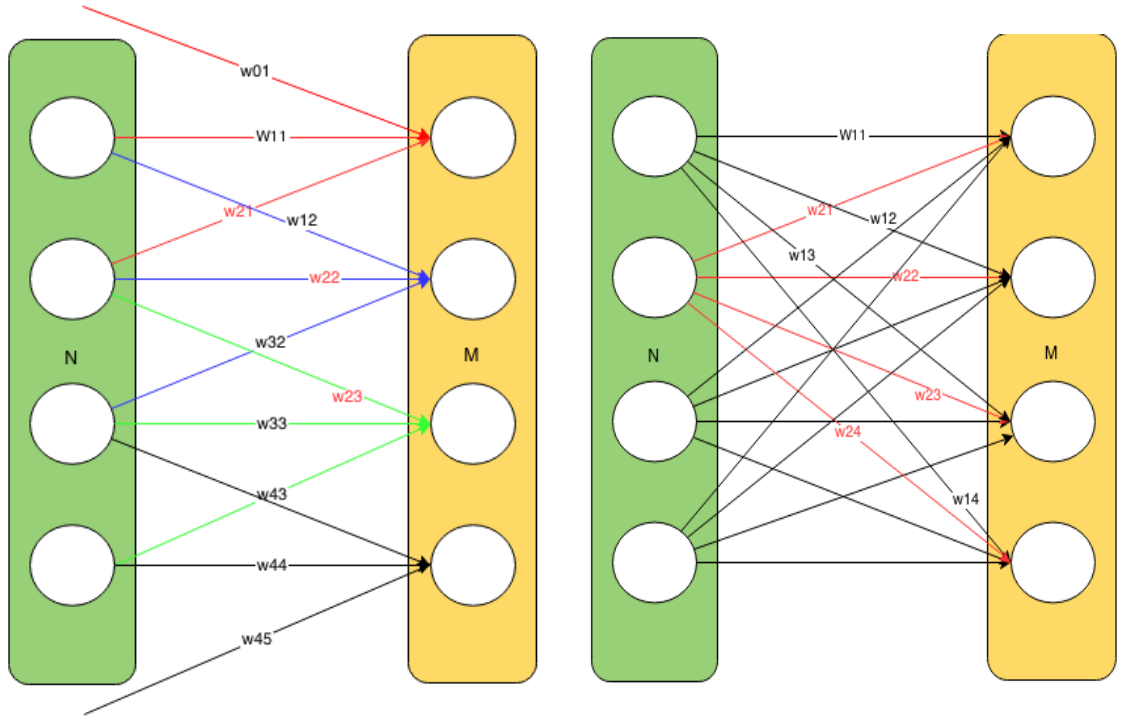
- Convolutional layer
- Nonlinearity
- Pooling layer
- Normalization layers
- Fully Connected layer
- Loss layer



Convolutional Neural Networks (CNN)

Types of layers

- Convolutional layer
- Nonlinearity
- Pooling layer
- Normalization layers
- Fully Connected layer
- Loss layer



Locally connected   Fully connected

Convolutional Neural Networks (CNN)

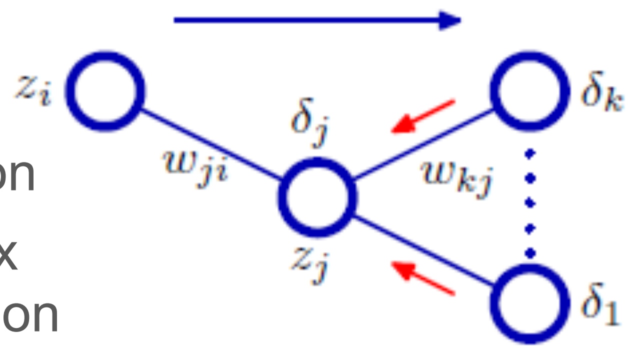
Types of layers

- Convolutional layer
- Nonlinearity
- Pooling layer
- Normalization layers
- Fully Connected layer
- Loss layer
  - Implements a loss function

Convolutional Neural Networks (CNN)

Learning

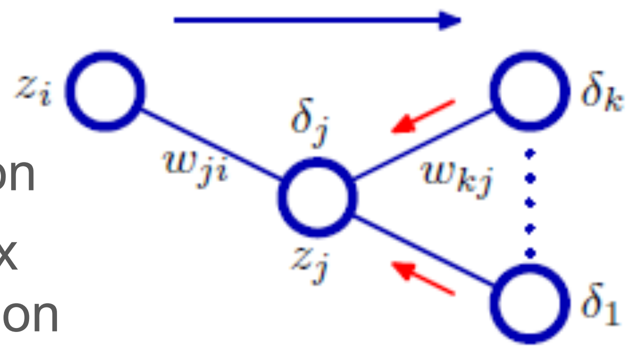
- Forward propagation for activation of all layers
- Backpropagation
  - Gradient of weights with respect to loss functions
    - **For regression**: Linear outputs and minimize the sum-of-squares function
    - **For binary classification**: We use a logistic sigmoid outputs and a cross-entropy error function
    - **For multiclass classification**: We use a softmax output and a multiclass cross-entropy error function



Convolutional Neural Networks (CNN)

Learning

- Forward propagation for activation of all layers
- Backpropagation
  - Gradient of weights with respect to loss functions
    - **For regression**: Linear outputs and minimize the sum-of-squares function
    - **For binary classification**: We use a logistic sigmoid outputs and a cross-entropy error function
    - **For multiclass classification**: We use a softmax output and a multiclass cross-entropy error function



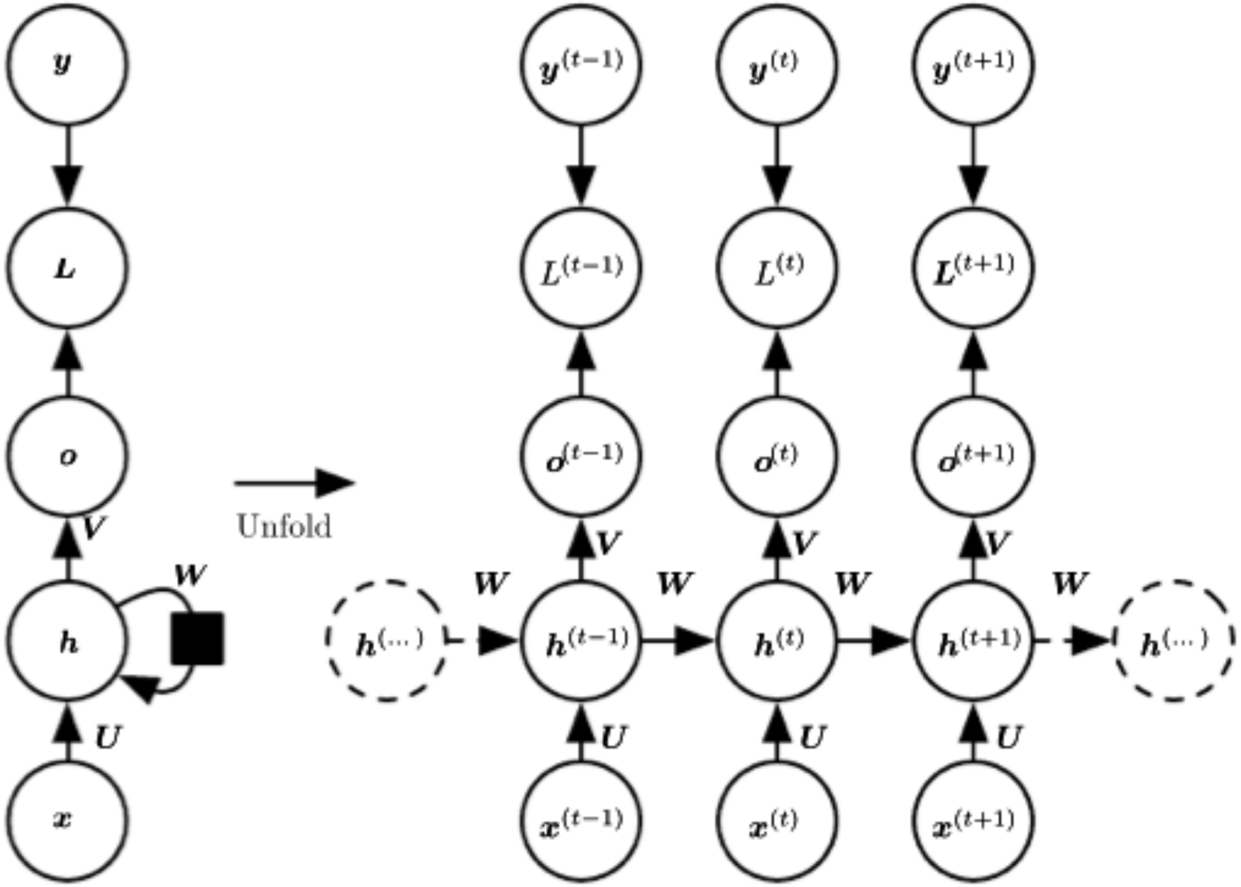
- https://cs.stanford.edu/people/karpathy/convnetjs/

Recurrent Neural Networks (RNN)

- Neural networks for processing sequential data $x^{(1)}, \ldots , x^{(T)}$
- Includes cycles that represent the influence of the present value of a variable on its own value at a future time step
- Also shares parameters across parts of the model
  - Shares the same weight across several time steps

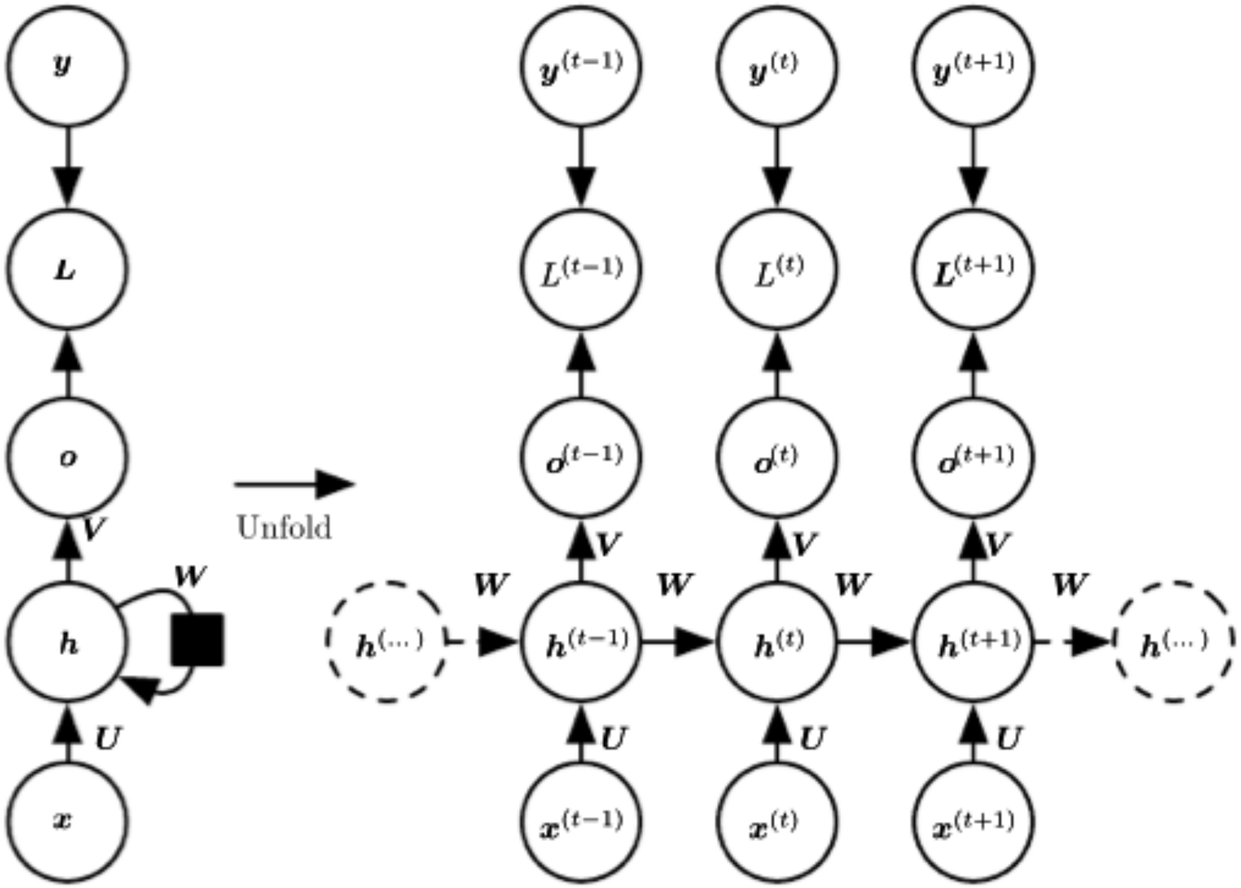
Recurrent Neural Networks (RNN)

- Recurrent networks that produce an output at each time step and have recurrent connections between hidden units



Recurrent Neural Networks (RNN)

- Backpropagation through time (BPTT)
  - Need to compute the Gradient



Deep neural networks

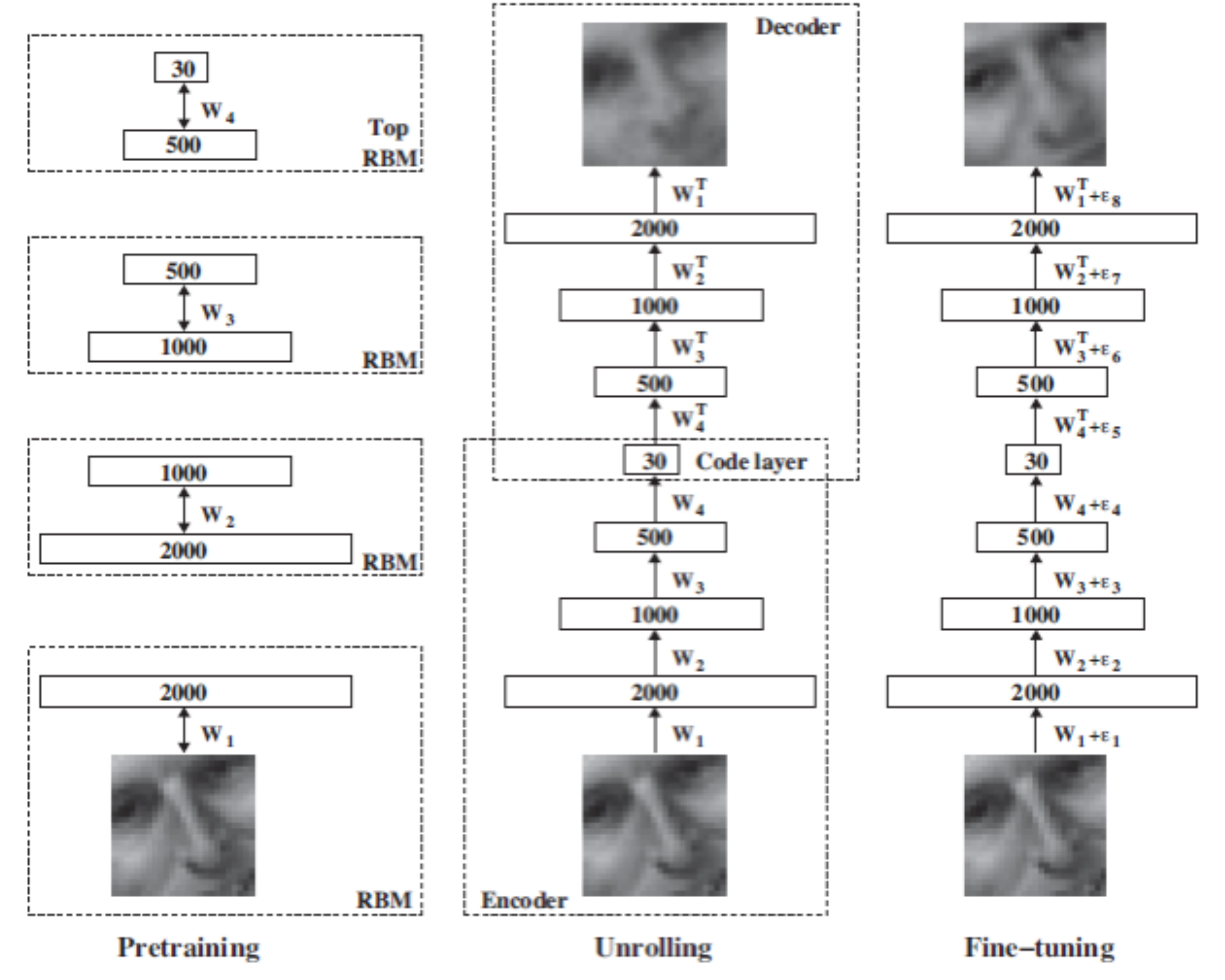
- Deep auto-enconders
  - A feedforward neural network that is trained to predict the input itself
  - To prevent the system from learning the trivial identity mapping, the hidden layer in the middle is usually constrained to be a narrow bottleneck

Deep neural networks

- Deep auto-enconders
  - A feedforward neural network that is trained to predict the input itself
  - To prevent the system from learning the trivial identity mapping, the hidden layer in the middle is usually constrained to be a narrow bottleneck
  - Usually pretrained by using an RBM
  - Fine-tuning with feedback

Deep neural networks

- Deep auto-enconders



Summary

Feed-forward networks

Training

Backpropagation

Deep Learning