# Introduction to Neural Networks

Machine Learning and Object Recognition 2015-2016

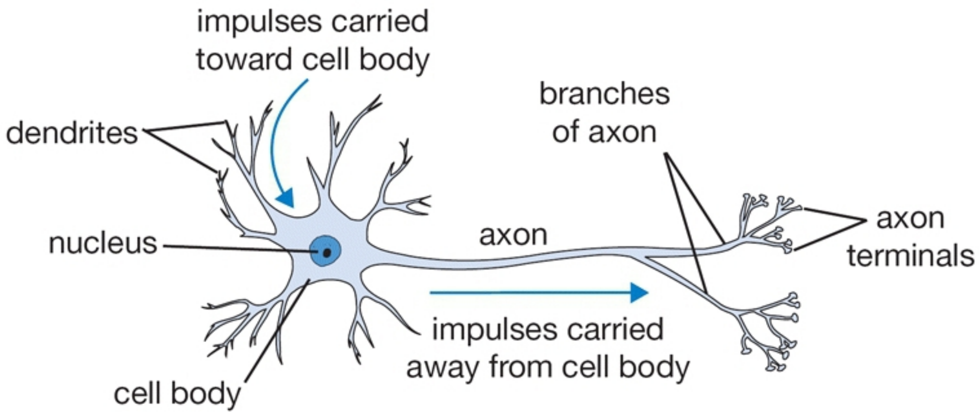
Jakob Verbeek, December 18, 2015

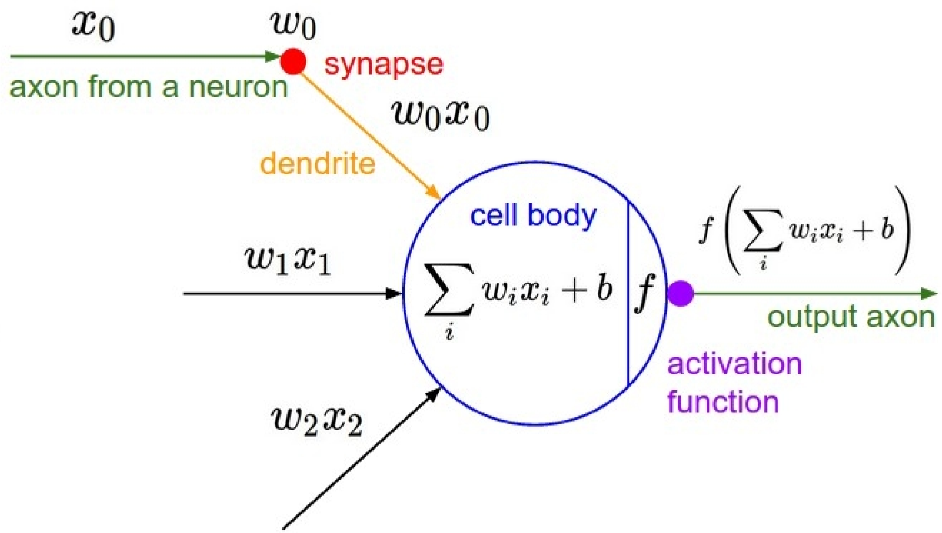
Course website:

http://lear.inrialpes.fr/~verbeek/MLOR.15.16

# Biological motivation

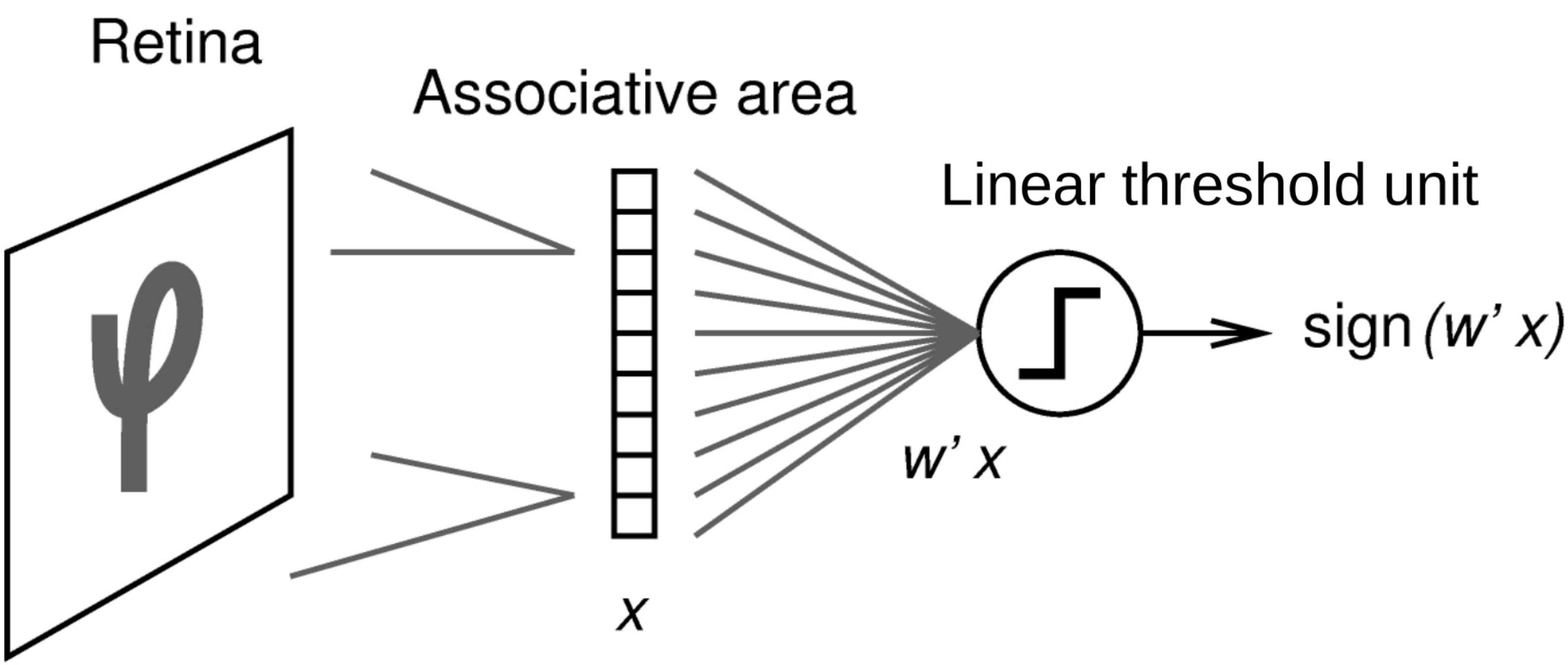
- Neuron is basic computational unit of the brain
  - about 10^11 neurons in human brain
- Simplified neuron model
  - Firing rate of electrical spikes is modeled as continuous quantity
  - Multiplicative interaction of input and connection strength (weight)
  - Multiple inputs accumulated in cell activation
  - Output if threshold activation is exceeded



# Rosenblatt's Perceptron

- One of the earliest works on artificial neural networks
  - First implementations in 1957 at Cornell University
  - Computational model of natural neural learning

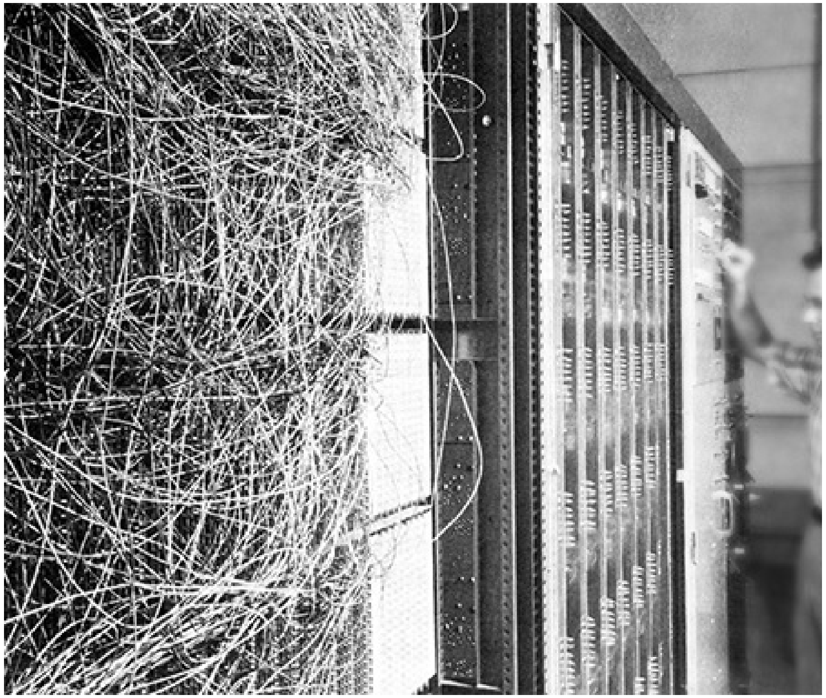


# Rosenblatt's Perceptron

- One of the earliest works on artificial neural networks
  - First implementations in 1957 at Cornell University
  - Computational model of natural neural learning
- Binary classification based on sign of generalized linear function

$$sign(f(x)) = sign(w^T \varphi(x))$$


20x20 pixel sensor



Random wiring on the computer

# Rosenblatt's Perceptron

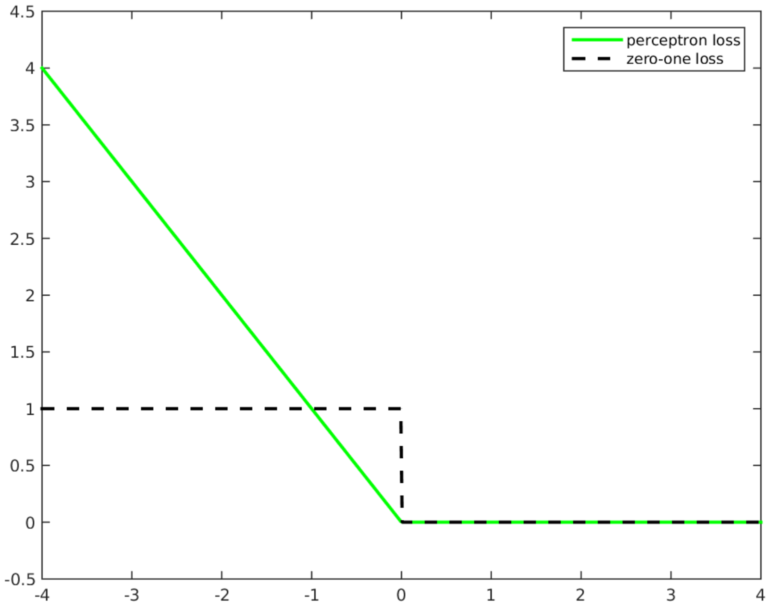
- Objective function linear in score over misclassified patterns

$$E(w) = -\sum_{t_i \neq sign(f(x_i))} t_i f(x_i) = \sum_i max(0, -t_i f(x_i))$$

- Perceptron learning via stochastic gradient descent

$$w^{t+1} = w^t + \eta \times t_i \varphi(x_i)[t_i f(x_i) < 0]$$

  - Eta is the learning rate




Potentiometers as weights, adjusted by motors during learning

# Limitations of the Perceptron

- Perceptron convergence theorem (Rosenblatt, 1962) states that
  - If training data is linearly separable
  - Then learning algorithm will find a solution in a finite number of iterations

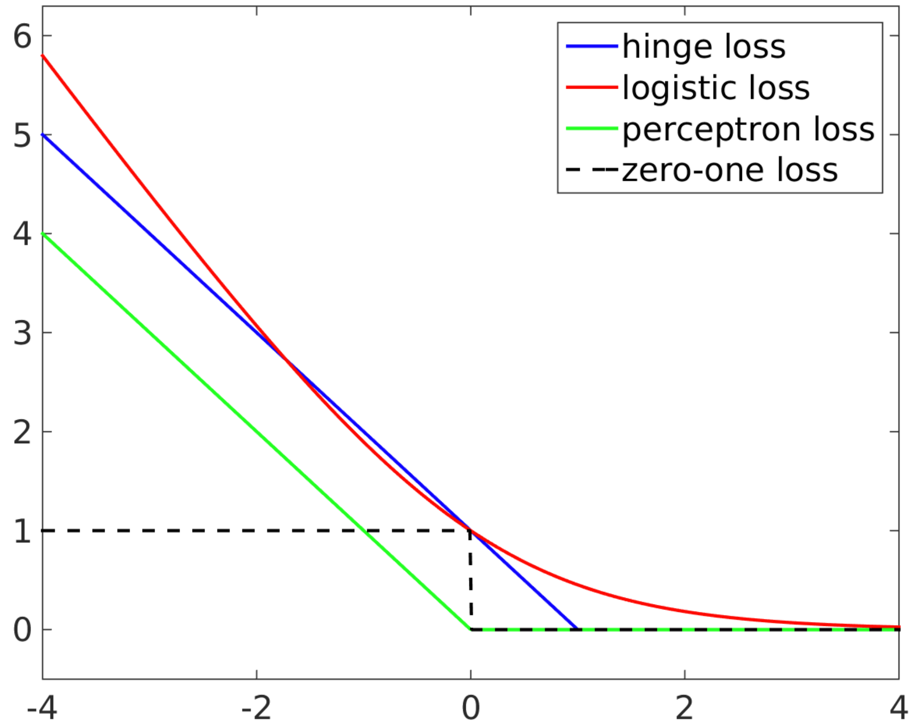
- If training data is linearly separable then the found solution will depend on the initialization and ordering of data in the updates
- If training data is not linearly separable, then the perceptron learning algorithm will never converge

- No direct multi-class extension
- No probabilistic output or confidence on classification

# Relation to SVM and logistic regression

- Perceptron similar to SVM without the notion of margin
  - Cost function is not a bound on the zero-one loss
- All are either based on linear function or generalized linear function by relying on pre-defined non-linear data transformation

$$f(x) = w^T \varphi(x)$$



# Classification with kernels

- Representer theorem states that in all these cases optimal weight vector is linear combination of training data

$$w=\sum_i \alpha_i \varphi(x_i)$$

$$f(x)=\sum_i \alpha_i \langle \varphi(x_i), \varphi(x) \rangle$$

- Kernel trick allows us (sometimes) to efficiently compute dot-products between high-dimensional transformations of the data

$$k(x_i, x)=\langle \varphi(x_i), \varphi(x) \rangle$$

  - Conversely, positive definite kernel functions compute dot-products between between possibly infinite dimensional data transformations

- Classification function is linear in data transformation given by kernel evaluations over the training data

$$f(x)=\sum_i \alpha_i k(x, x_i)=\alpha^T k(x, .)$$

# Limitation of kernels

- Classification based on weighted similarity to training samples
  - Design of kernel based on domain knowledge and experimentation

$$f(x)=\sum_i \alpha_i k(x, x_i)=\alpha^T k(x,.)$$

  - Some kernels are data adaptive, for example the Fisher kernel

- Number of free variables grows linearly in the size of the training data

- Alternatively: fix the number of “basis functions” in advance
  - Choose a family of non-linear basis functions
  - Learn the parameters, together with those of linear function

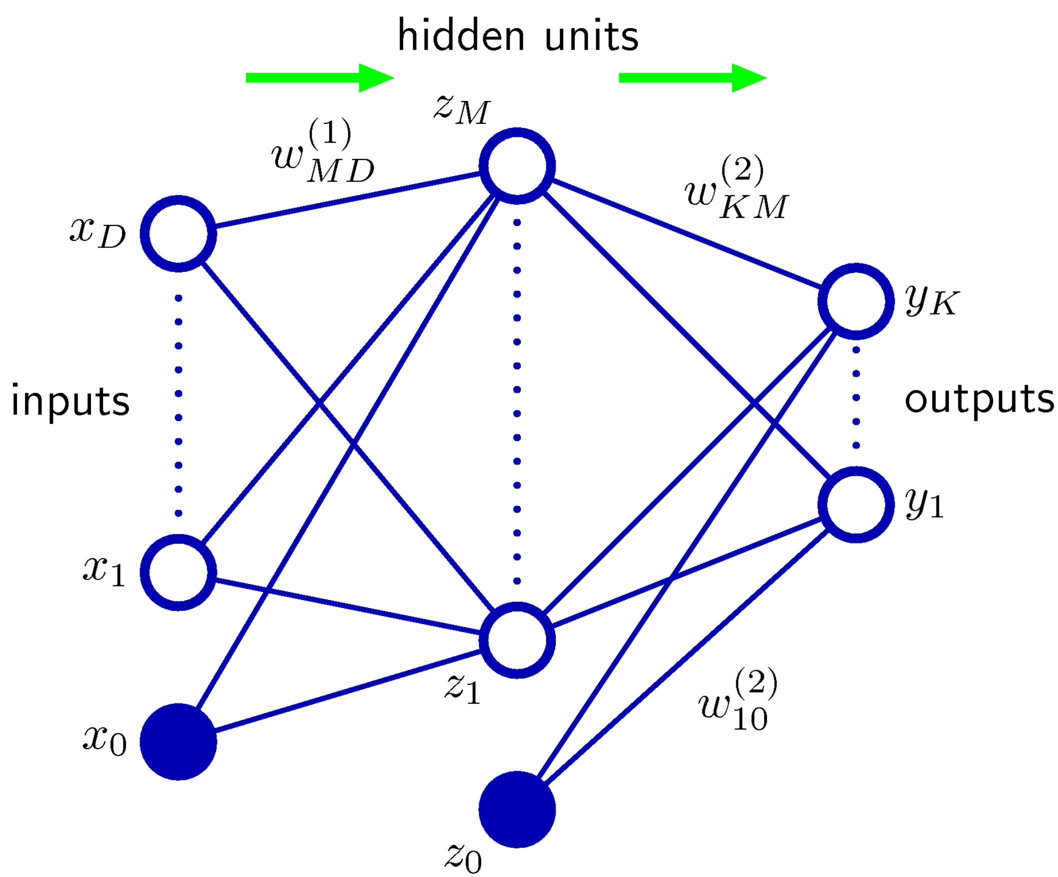
$$f(x)=\sum_i \alpha_i \varphi_i(x;\theta_i)$$

# Feed-forward neural networks

- Define outputs of one layer as scalar non-linearity of linear function of input
- Known as “multi-layer perceptron”
  - Perceptron has a step non-linearity of linear function
  - Other non-linearities used in practice

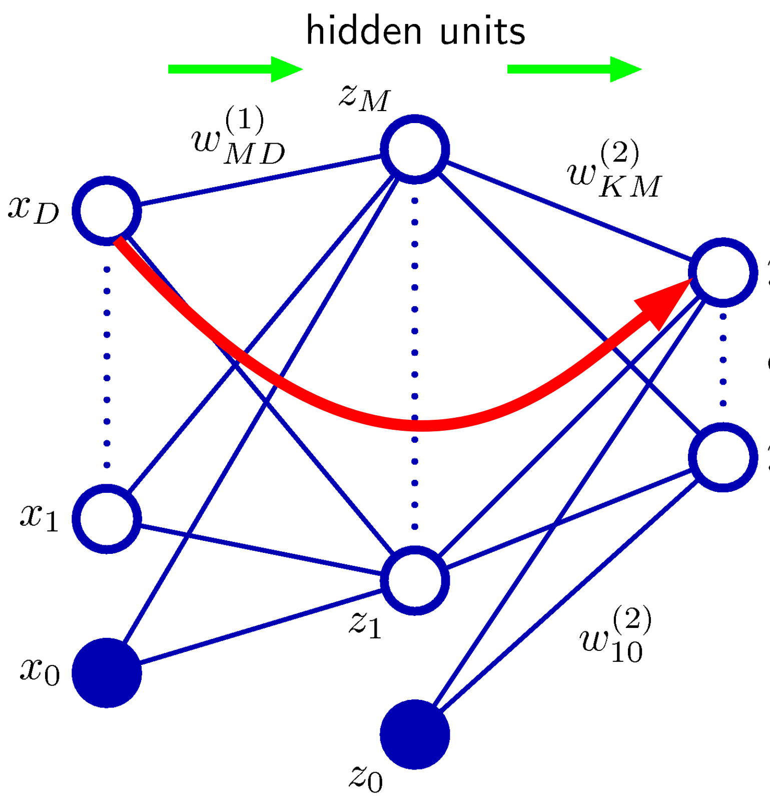
$$z_j = h(x^T w_j^{(1)})$$

$$y_k = \sigma(z^T w_k^{(2)})$$



# Feed-forward neural networks

- If "hidden layer" activation function is taken to be linear than a single-layer linear model is obtained
- Two-layer networks with linear outputs can uniformly approximate any continuous function on a compact input domain to arbitrary accuracy provided the network has a sufficiently large number of hidden units
  - Holds for many non-linearities, but not for polynomials
- Architecture can be generalized
  - More than two layers of computation
  - Skip-connections from previous layers
  - Directed acyclic graphs of connections
- Key difficulties
  - How design the network architecture
    - Nr nodes, layers, non-linearities,
  - Learn the optimal parameters
    - Non-convex optimization



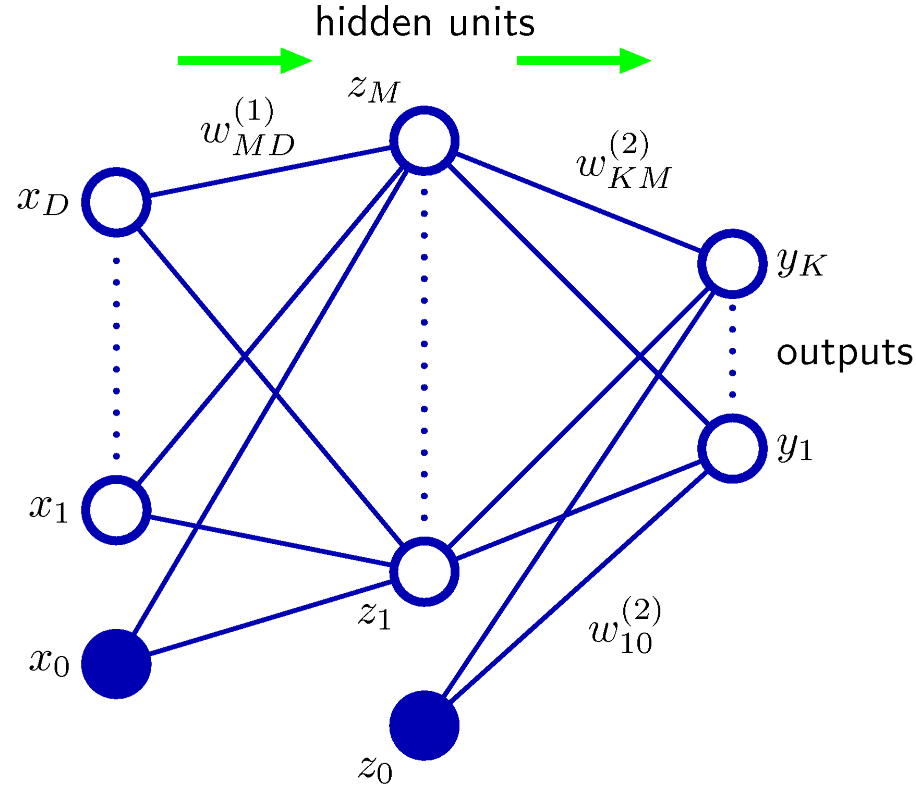
# Multi-class classifiction

- One output score for each target class
- Multi-class logistic regression loss
  - Define probability of classes by softmax over scores
  - Maximize log-probability of correct class

$$p(y=c|x)=\frac{\exp y_c}{\sum_k \exp y_k}$$

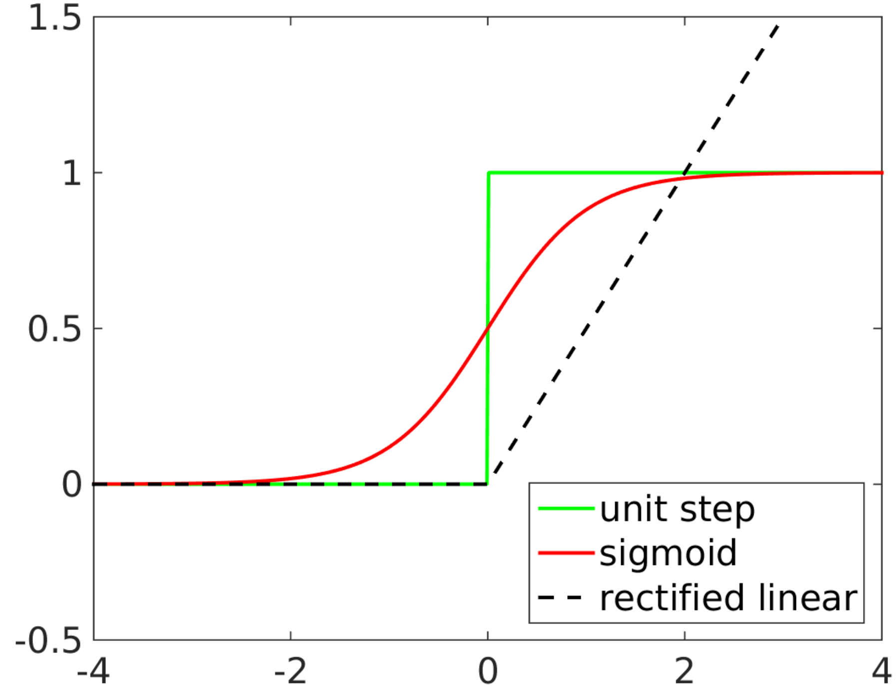
- Precisely as before, only difference is that we are now learning the data transformation concurrently with the classifier

- Representation learning in discriminative and coherent manner
- Fisher kernel also data adaptive but not discriminative and task dependent
- More generally, we can choose a loss function for the problem of interest and optimize all network parameters w.r.t. this objective (regression, metric learning, ...)



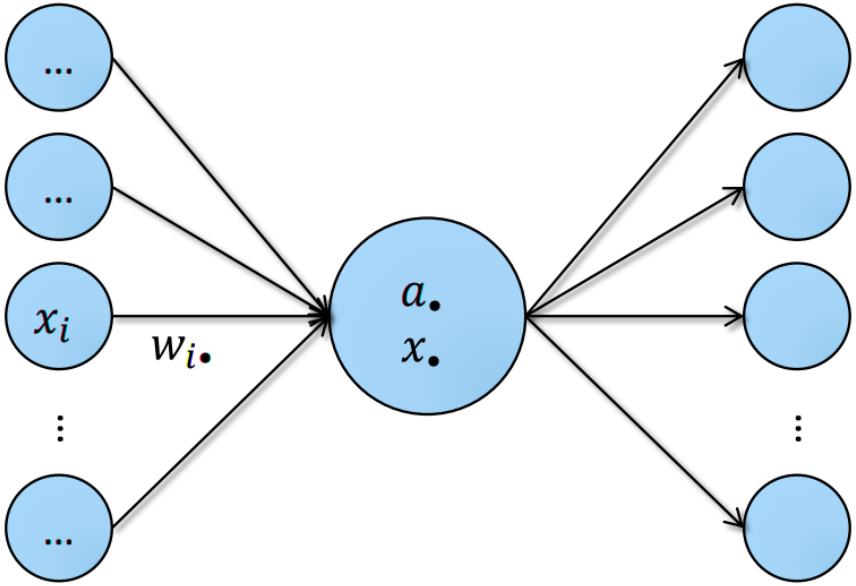
# Activation functions

- Unit step function, used in original Perceptron
  - Discontinuous, not possible to propagate error
- Sigmoid function: Smooth step function
  - Gradients saturate except in transition regime
  - Hyperbolic tangent: same but zero-centered instead
- Rectified linear unit (ReLU): Clips negative values to zero
  - One-sided saturation only, very cheap to compute
- Max-out: max of two linear functions
  - Similar as ReLU
  - No constant regimes at all



# Training the network: forward and backward propagation

- Forward propagation from input nodes to output nodes
  - Accumulate inputs into weighted sum
  - Apply scalar non-linear activation function f
- Use Pre(j) to denote all nodes feeding into j

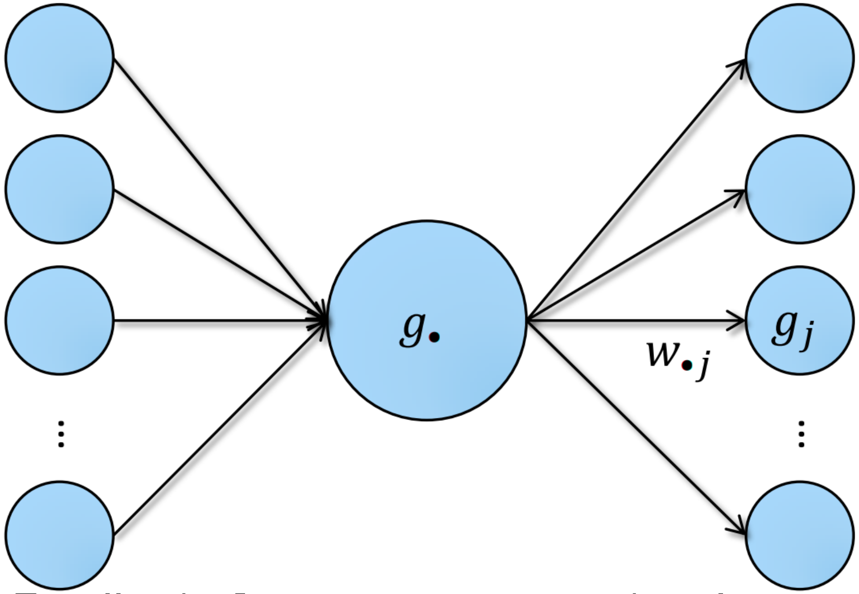



$$a_j = \sum_{i \in Pre(j)} w_{ij} x_i$$

$$x_j = f(a_j)$$

# Training the network: forward and backward propagation

- Backward propagation of loss gradient from output nodes to input nodes
  - Application of chainrule of derivatives
  - Accumulate gradients from downstream nodes
  - Multiply with derivative of local activation
- Use Post(i) to denote all nodes that i feeds into




$$g_i = \frac{\partial L}{\partial a_i}$$

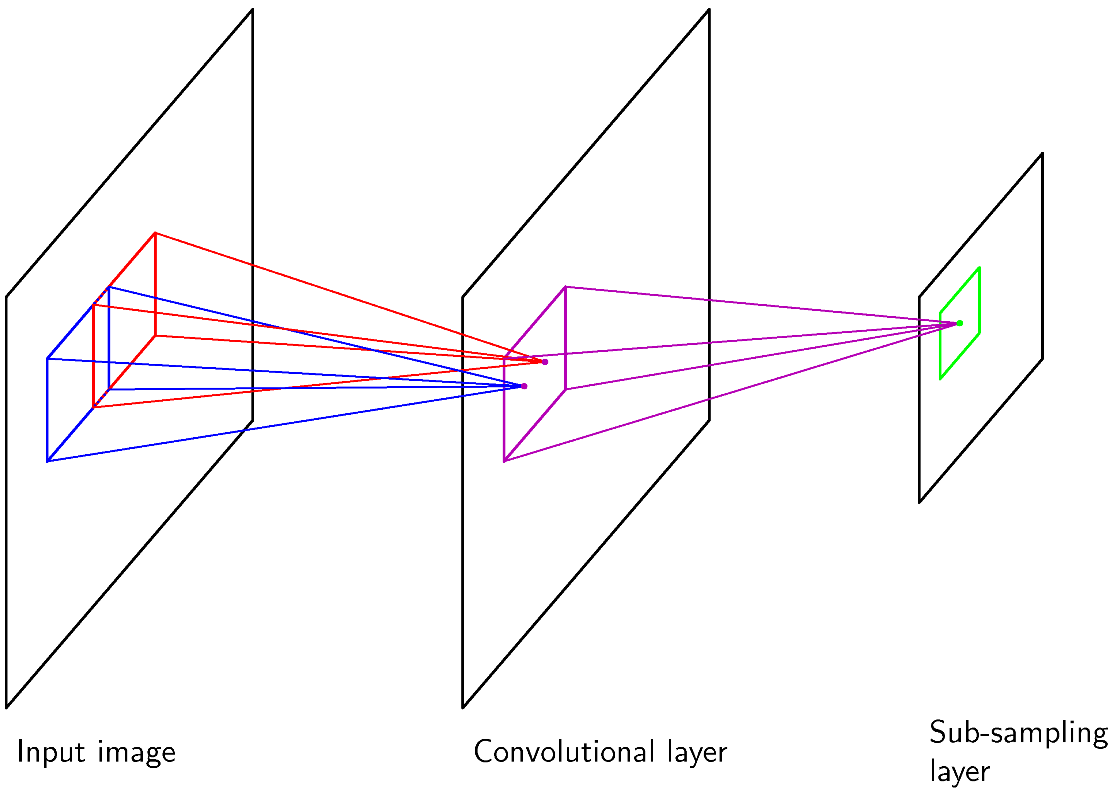
$$g_i = f'(a_i) \sum_{j \in Post(i)} w_{ij} g_j$$

$$\frac{\partial L}{\partial w_{ij}} = x_i g_j$$

- Gradient of weights between two layers given by outer-product of x and g

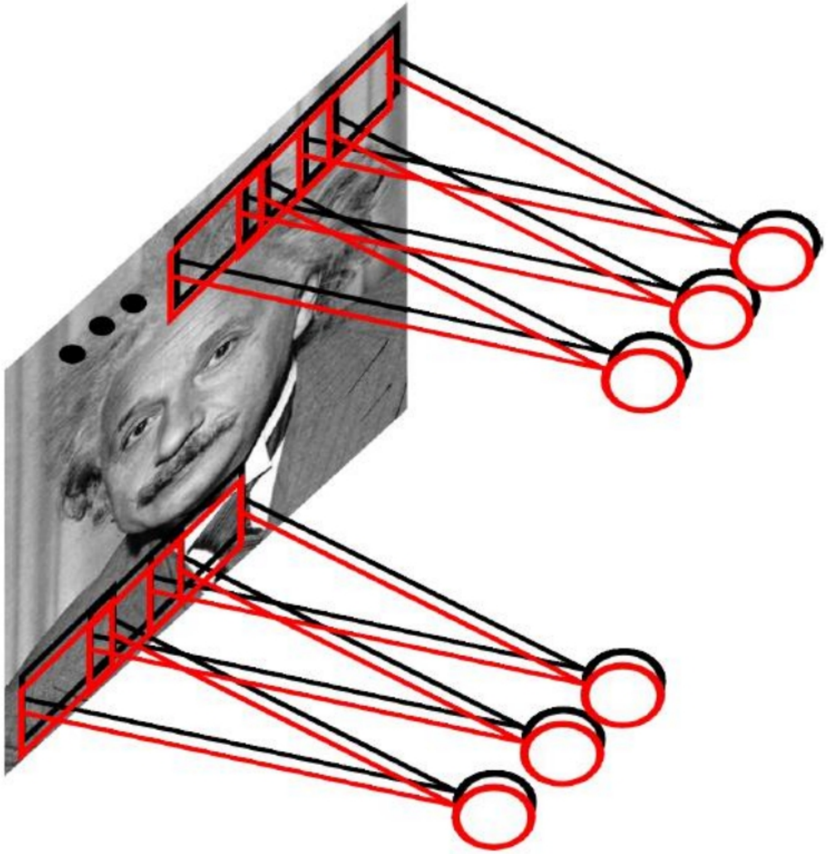
# Convolutional neural networks

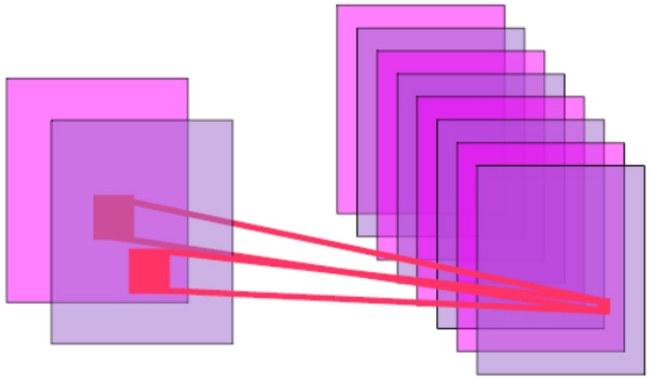
- Local connections: motivation from findings in early vision
  - Simple cells detect local features
  - Complex cells pool simple cells in retinotopic region
- Convolutions: motivated by translation invariance
  - Same processing should be useful in different image regions



# Convolutional neural networks

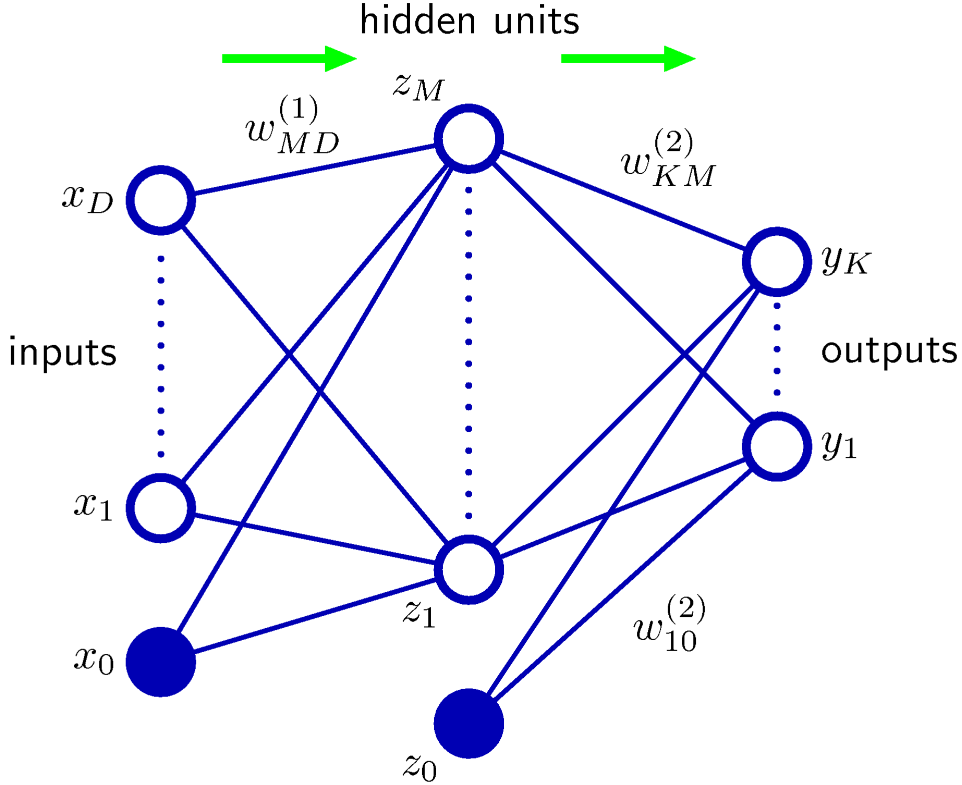
- Multiple convolutions per layer
  - Different features
  - Same level of abstraction and scale

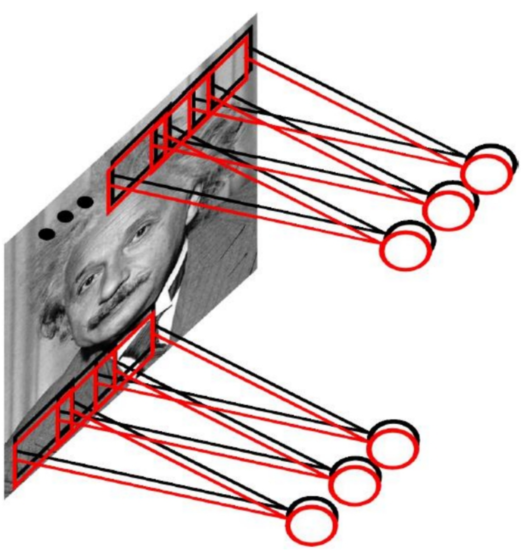


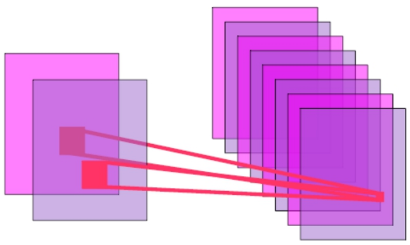


# Relation to “fully connected” neural networks

- Hidden units
  - Spatially organized: output of convolution filter at certain position
  - Local connectivity: depend only on small fraction of input units
- Connection weights
  - Same filter weights for an output map
  - Massive weight sharing: nr. of parameters does not grow in output size

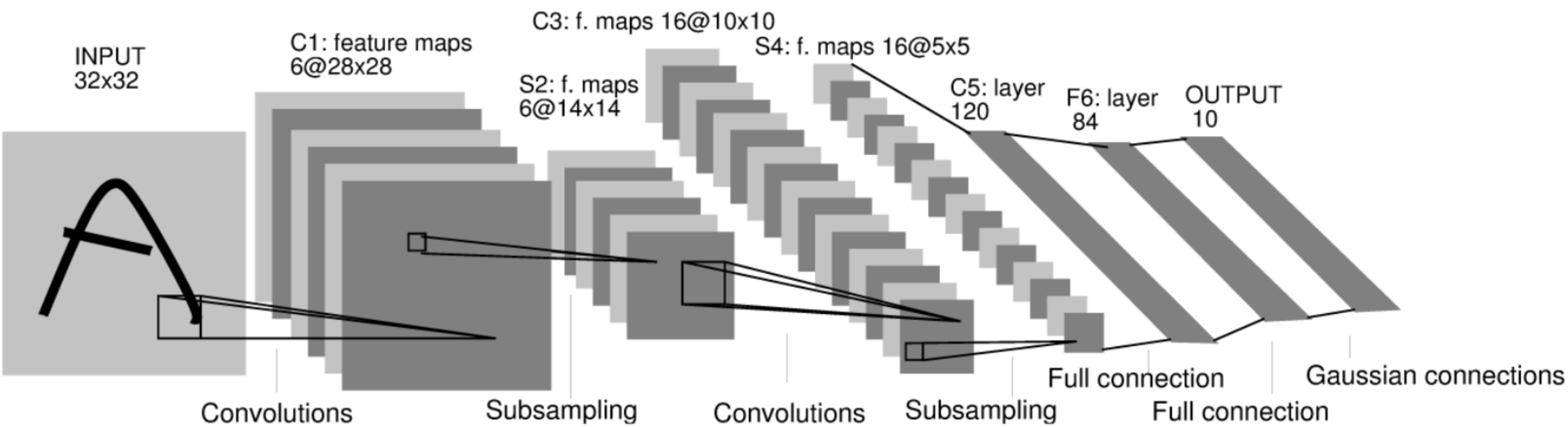




# Convolutional neural network architectures

- Convolutional layers: local features along scale and abstraction hierarchy
  - Convolution
  - Nonlinearity
  - Pooling, eg. max response in small region
- Fully connected layers: assemble local features into global interpretation
  - Multi-layer perceptron

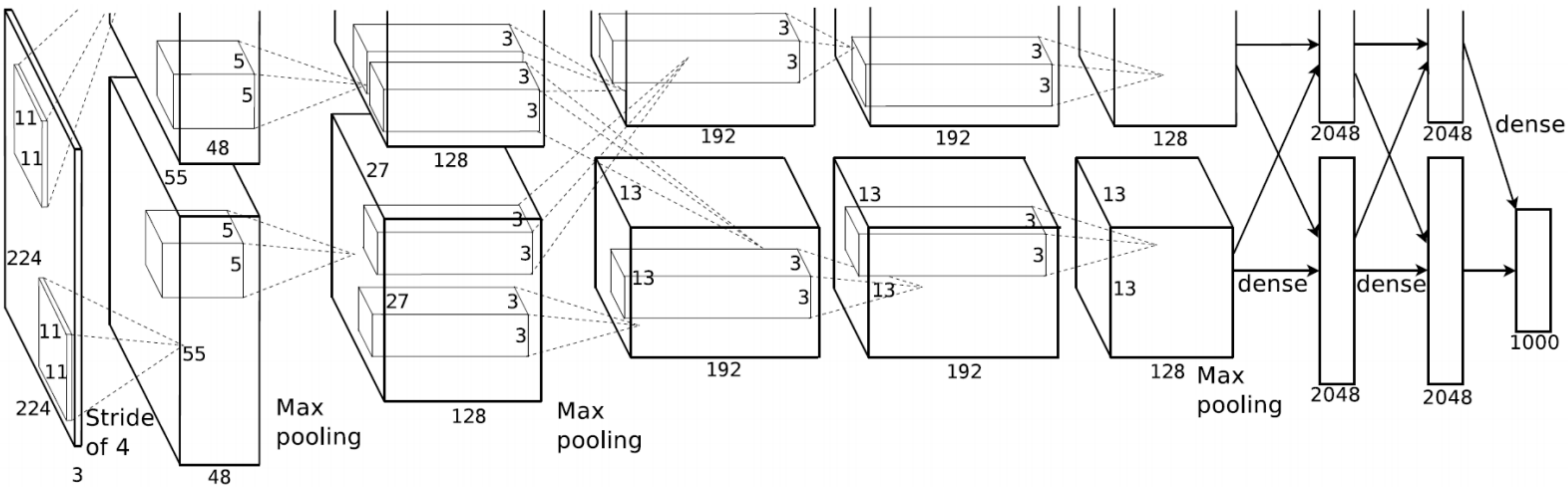



Handwritten digit recognition.

LeCun, Bottou, Bengio, Haffner, Proceedings IEEE, 1998

# Convolutional neural network architectures

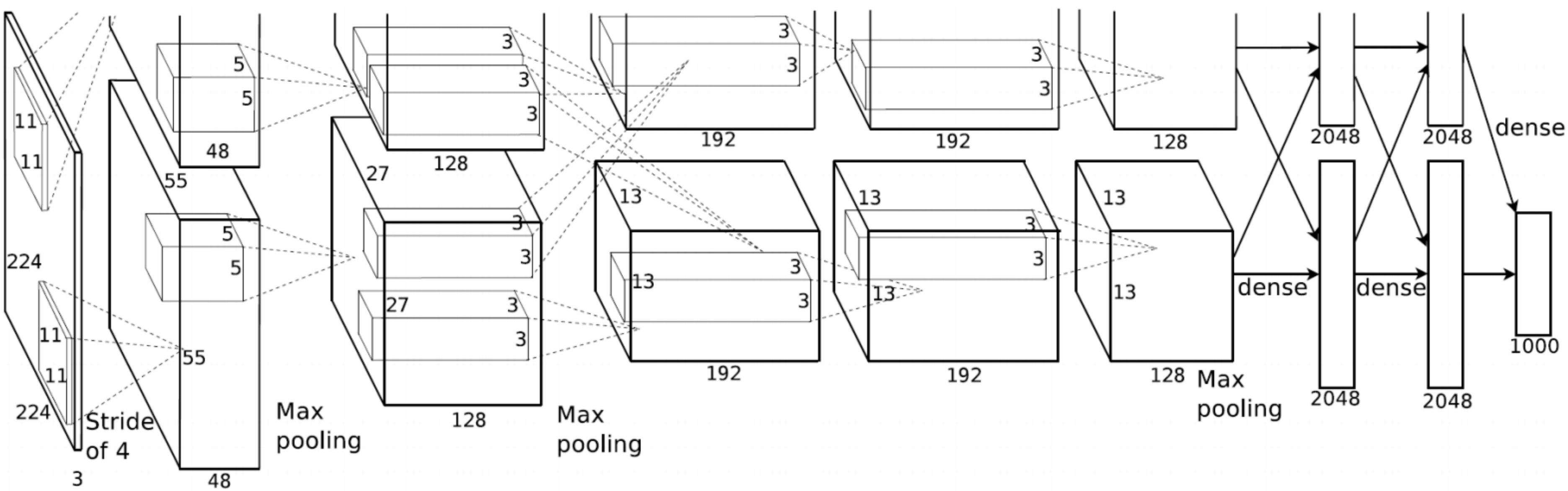
- Similar architectures for general object recognition a decade later
- Deeper: e.g. 19 layers in Simonyan & Zisserman, ICLR 2015
- Wider: More filters per layer: hundreds instead of tens
- Wider: thousands of nodes in fully connect layers
- ReLU activations instead of hyperbolic tangent




Krizhevsky & Hinton, NIPS 2012, Winning model ImageNet 2012 challenge

# Convolutional neural network architectures

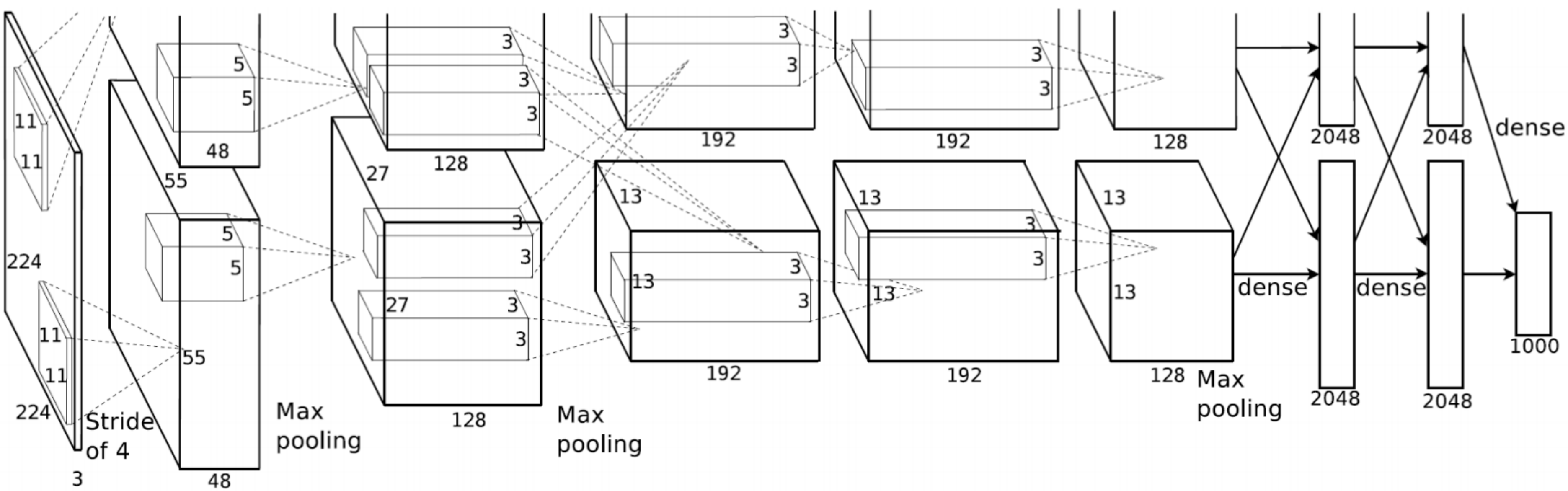
- Similar architectures for general object recognition a decade later
- More training data
  - 1.2 millions of 1000 classes for ImageNet challenge
  - 200 million faces in Schroff et al, CVPR 2015
- GPU-based implementations
  - Massively parallel computation of convolutions
  - Krizhevsky & Hinton, 2012: six days of training on two GPUs




Krizhevsky & Hinton, NIPS 2012, Winning model ImageNet 2012 challenge

# Understanding convolutional neural network activations

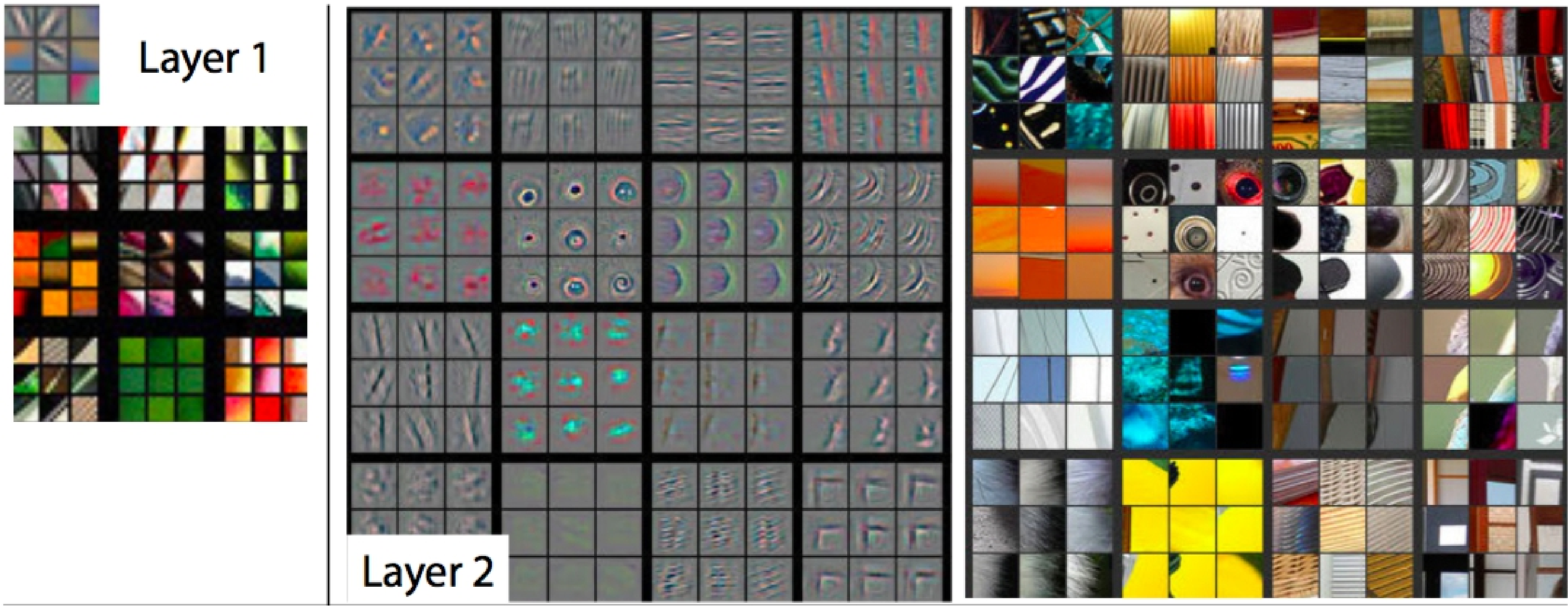
- Architecture consists of
  - 5 convolutional layers
  - 2 fully connected layers
- Visualization of patches that yield maximum response for certain units
  - We will look at each of the 5 convolutional layers




Krizhevsky & Hinton, NIPS 2012, Winning model ImageNet 2012 challenge

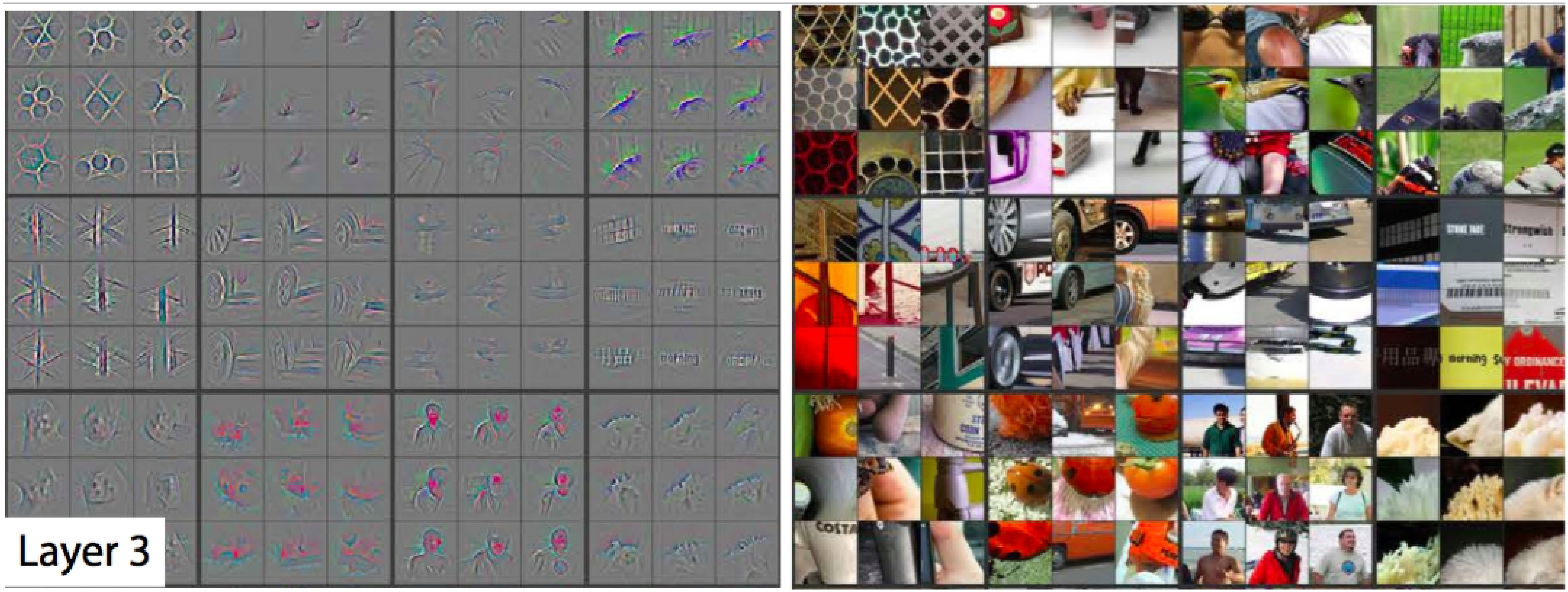
# Understanding convolutional neural network activations

- Layer 1: simple edges and color detectors
- Layer 2: corners, center-surround, ...



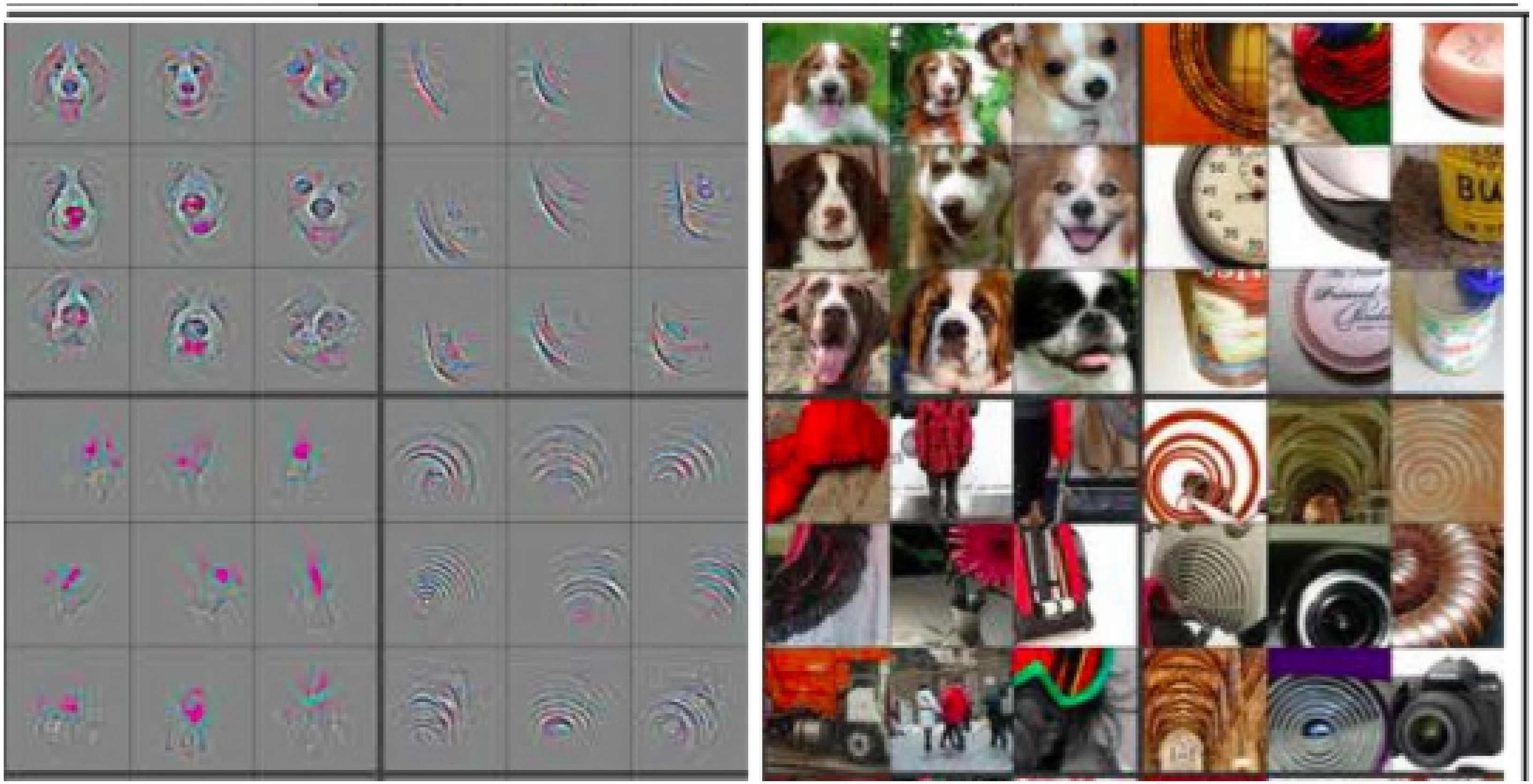
# Understanding convolutional neural network activations

- Layer 3: textures, object parts



# Understanding convolutional neural network activations

- Layer 4: complex textures and object parts



# Understanding convolutional neural network activations

- Layer 5: complex textures and object parts