Introduction to Big Data and Machine Learning
Neural Networks
Slides primarily based on Ch 5. PRML book by Bishop

Dr. Mihail

November 5, 2019

Neural Networks

History

- Origins in attempts to find mathematical representations of information processing in biological systems (cca. 1943 McCulloch and Pitts)
- In essence: input goes through a sequence of transformations using a fixed number of basis functions
- Many (more each day) variations exist, that are domain specific (e.g., convolutional neural networks, recurrent neural networks, etc.)

Neural Networks

Feedforward Neural Networks

- We begin with a reminder of models for generalized linear regression and classification, based on linear combinations of fixed non-linear basis functions $\phi_j(x)$

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

  where f is a nonlinear activation function in case of classification, and is the identity in case of regression

- Neural networks extend this model by making the basis functions $\phi_j(x)$ depend on parameters and to allow these parameters to be adjusted, along with the coefficients $\{w_j\}$ during training

Basic Neural Networks

Basics

- A series of functional transformations
- First, we construct M linear combinations of the input variables $x_1, \ldots, x_D$ in the form

$$a_j = \sum_{i=1}^{D} w_{j0}^{(1)} x_i + w_{j0}^{(1)} \tag{2}$$

  where $j = 1, \ldots, M$ and the superscript (1) indicates that the corresponding parameters are in the first "layer" of the network
- We shall refer to parameters $w_{ji}^{(1)}$ as *weights* and the parameters $w_{j0}^{(1)}$ as *biases*

Neural Networks

Basics

- The parameters a_j are known as *activations*, each of them transformed using a differentiable, nonlinear activation function h to give

$$z_j = h(a_j) \tag{3}$$

- These quantities correspond to the outputs of the basis functions ϕ, where in the context of NNs are called *hidden units*
- The nonlinear funcions h are chosen based on domain specific applications (e.g., ReLU, sigmoid, tanh), whose values are combined linearly to give *output unit activations*

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \tag{4}$$

where $k = 1, \ldots, K$ and K is the total number of outputs

Neural Networks

Output

- Finally, the output unit activations are transformed using an appropriate activation function, to give a set of network outputs y_k
- For standard regression problems, the activation can be the identity $y_k = a_k$
- For binary classification problems, each output is transformed using a logistic sigmoid function

$$y_k = \sigma(a_k) \tag{5}$$

where

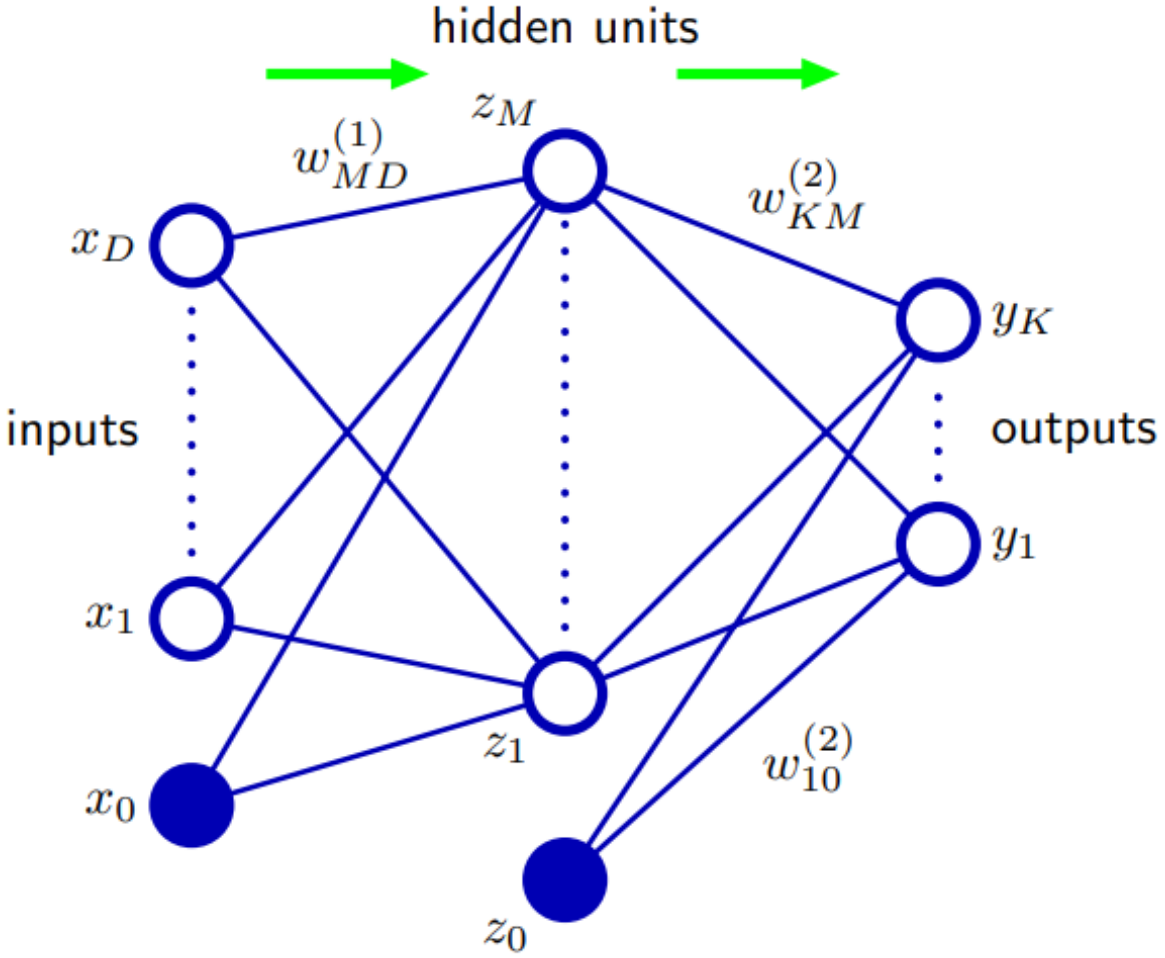
$$\sigma(a) = \frac{1}{1 + exp(-a)} \tag{6}$$

- For multiclass problems, softmax can be used

Neural Networks

Example

Network diagram for the two-layer neural network corresponding to (5.7). The input, hidden, and output variables are represented by nodes, and the weight parameters are represented by links between the nodes, in which the bias parameters are denoted by links coming from additional input and hidden variables x_0 and z_0. Arrows denote the direction of information flow through the network during forward propagation.



Neural Networks

Model

- We can combine these various stages to give the overall network:

$$y_k(x, w) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right) \tag{7}$$

- The NN model is simply a nonlinear function from a set of input variables $\{x_i\}$ to a set of output variables $\{y_k\}$ controlled by a vector w of adjustable parameters
- The function shown in the figure in the previous slide can then be interpreted as a forward propagation of information through the network

Neural Networks

Nomenclature

- These models first came to be known as multilayer perceptron (MLP) due to multiple uses of nonlinearities through the layers
- The “deep” models simply refer that there are many such layers of nonlinearities
- These functions are differentiable w.r.t. network parameters, a key important fact for training

Neural Networks

Number of internal (hidden) nodes

- If activations are linear, and the number of internal nodes are greater than both input and output nodes, there is an equivalent linear function
- If the activations are linear, and the number of hidden units are less than the input or the output nodes, this can be shown to be related to PCA dimensionality reduction

Training Neural Networks

Training

- NNs are a general class of parametric, nonlinear functions from a vector x to a vector y (or t for target, as used in previous lectures) of output variables
- One way to think about learning, is to think of "fitting" the model from the inputs x to the targets t, by minimizing some error function:

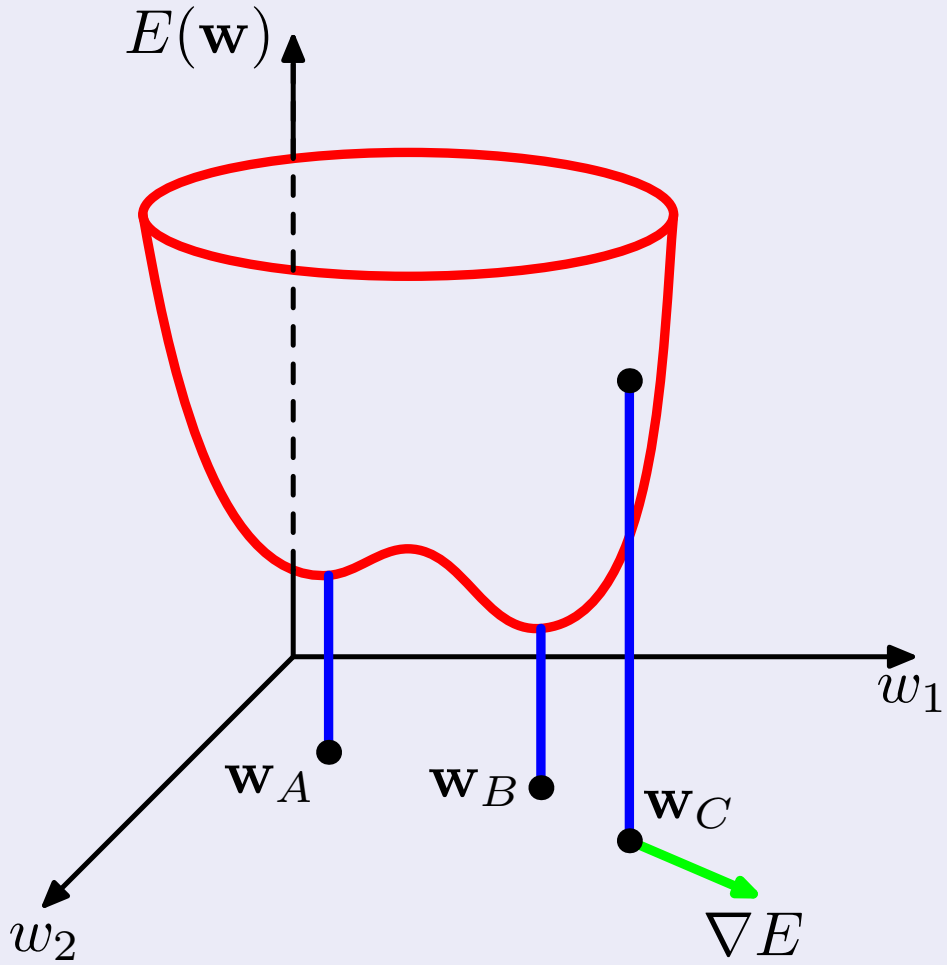
$$E(w) = \frac{1}{2} \sum_{n=1}^{N} \|y(x_n, w) - t_n\|^2 \tag{8}$$

- Many ways to optimize the above function, but $|w|$ in modern networks can be millions

Neural Networks

Optimizing E

- The task: find w that minimizes $E(w)$
- Looking at this geometrically, w define an error surface



Neural Networks

Gradually minimize

- Take small steps in the weight space from w to $w + \delta w$
- When doing so, $\delta E \approx \delta w^T \nabla E(w)$, where ∇E is a vector that points in the direction of the greatest rate of increase of E
- Smallest E will occur where the gradient vanishes

$$\nabla E(w) = 0 \tag{9}$$

- Analytic solution is not feasible (network with M hidden units, each point in weight space is a member of a family of $M!2^M$ equivalent points)
- Compromise: can take small steps in the direction of $-\nabla E(W)$

Neural Networks

Gradient descent algorithm

- Initialize with some random $w^{(0)}$
- Iterate: $w^{(\tau+1)} = w^{(\tau)} + \Delta w^{(\tau)}$ until convergence
- MANY algorithms (solvers) exist to do this efficiently and find good solutions
- Outside of lecture scope: analytical computation of $\frac{\partial E_n}{\partial w_{ji}}$, done by repeated application of chain rule

Neural Networks

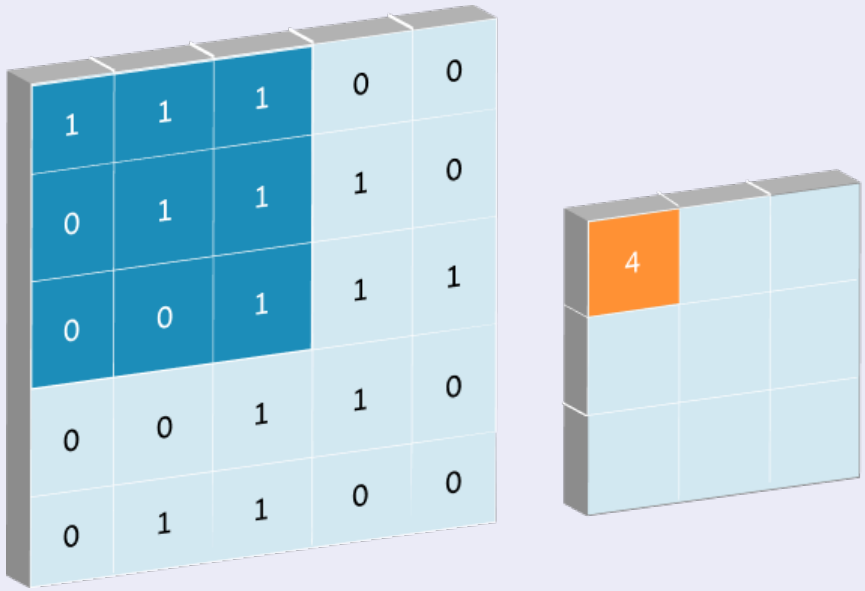
Convolutional Neural Networks

- A special type of feedforward network architecture, where the input is an image (matrix or tensor, for color images)
- Why? To achieve invariance typical of imagery (e.g., translation, rotation, scale)

Neural Networks

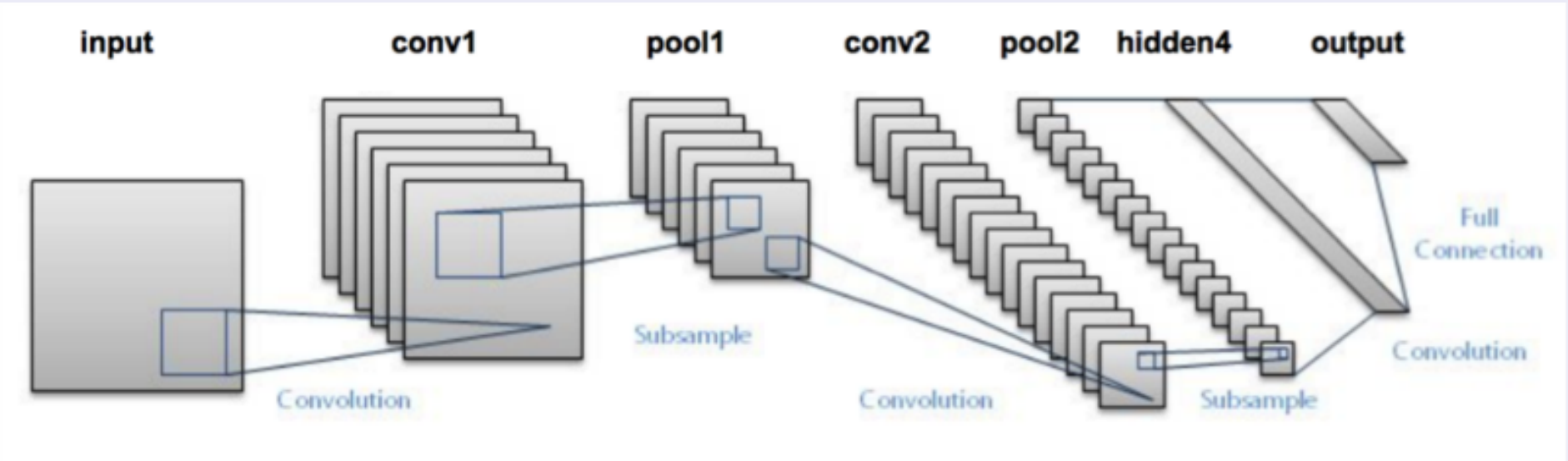
Convolution in CNNs

- Simply a dot product, element wise multiplication of a small matrix (kernel) with part of an image



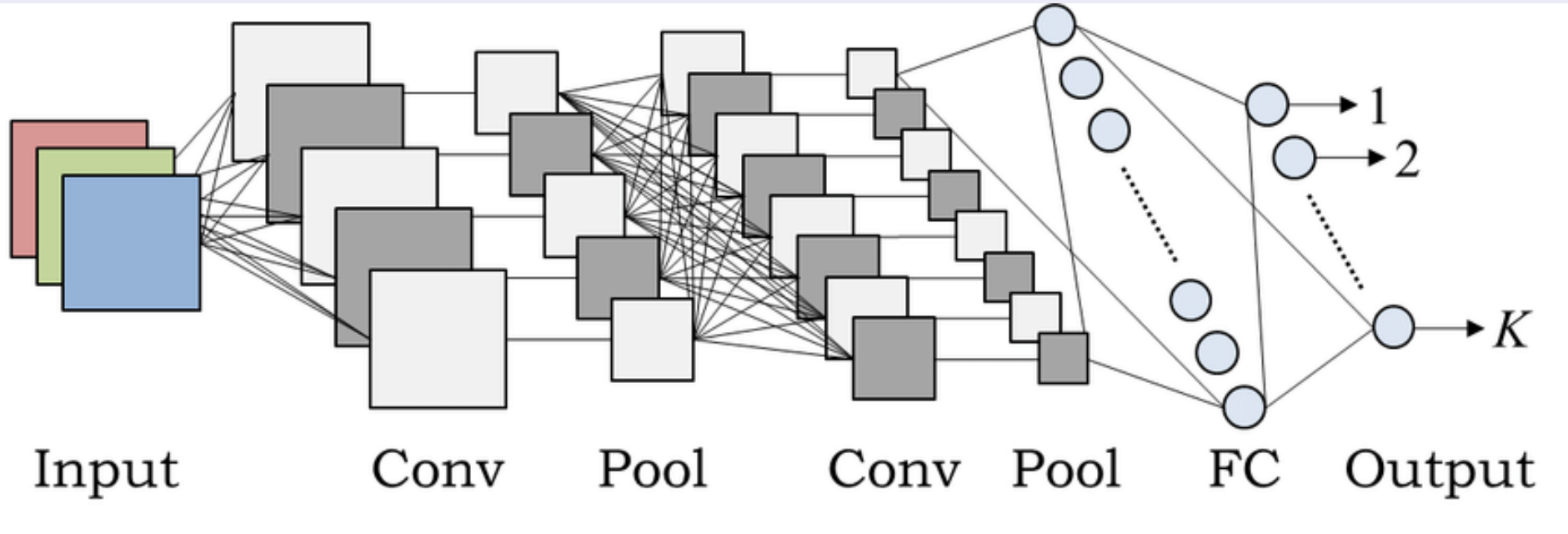
Neural Networks

CNN example



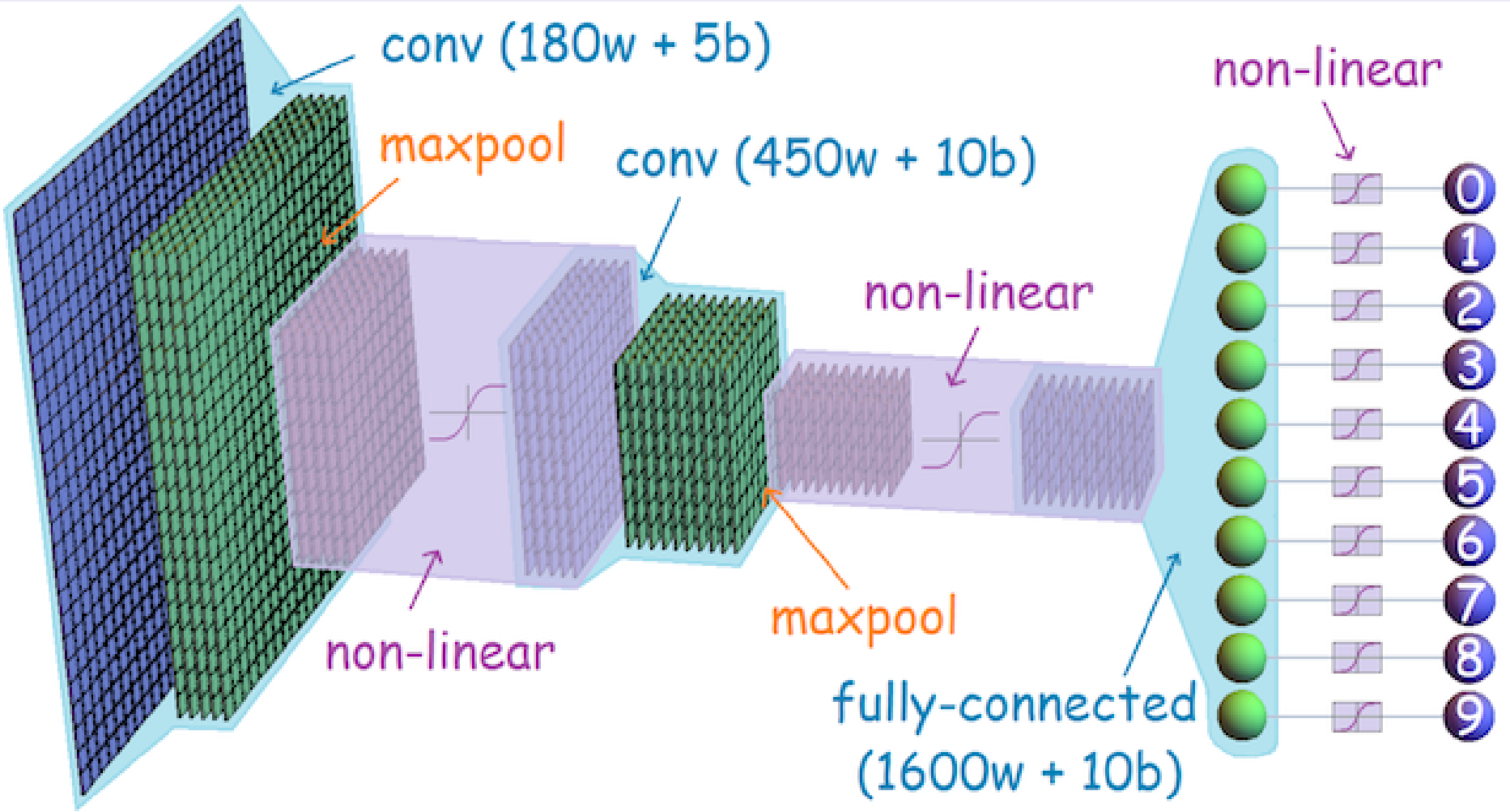
Neural Networks

Another CNN example



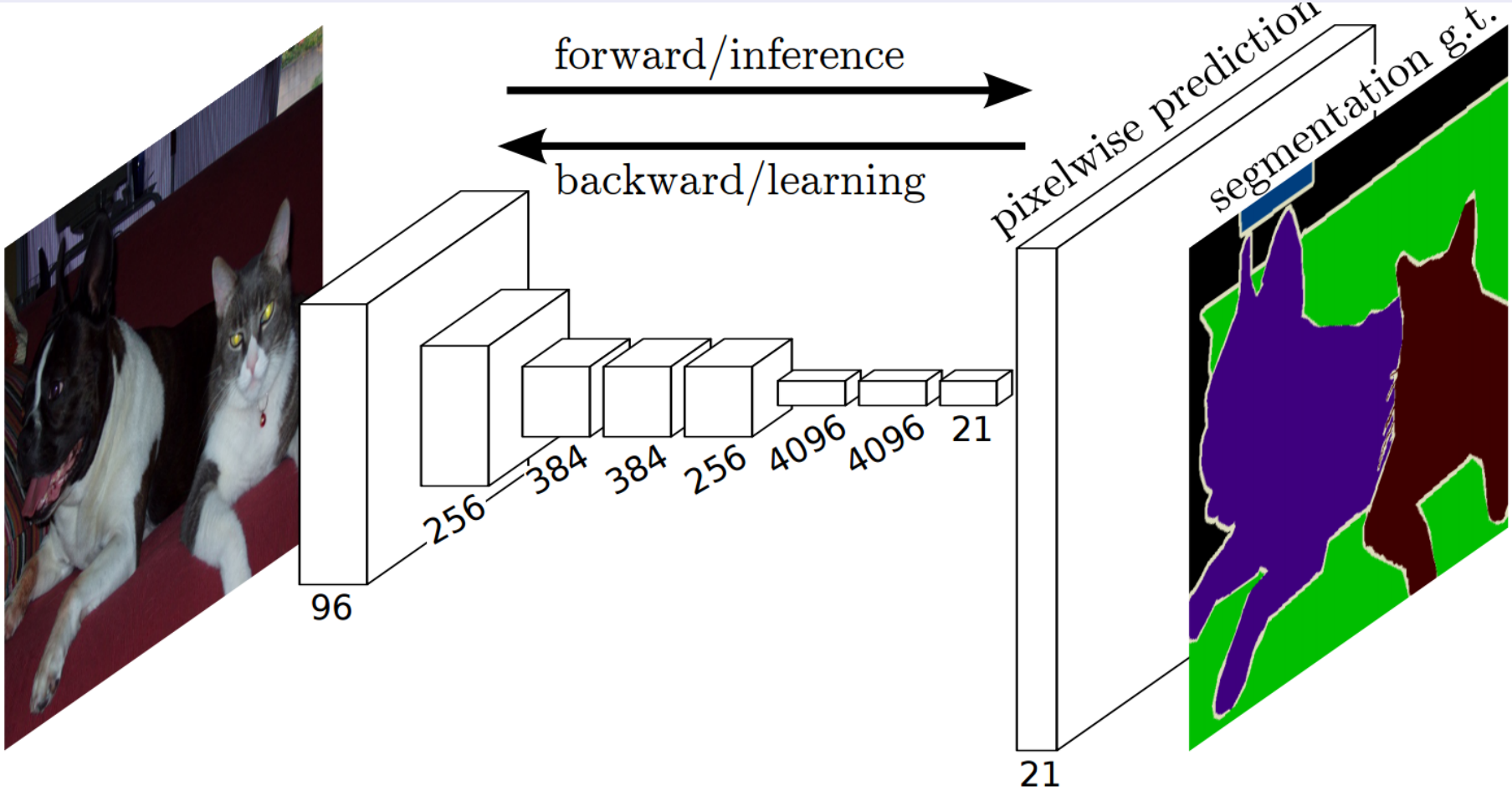
Neural Networks

Another CNN example



Fully convolutional Neural Networks

Fully Convolutional NN



Neural Networks

Implementations

- Tens of popular ones, most widely known:
  1. Tensorflow
  2. Keras
  3. Caffee
  4. Torch
  5. DeepPy