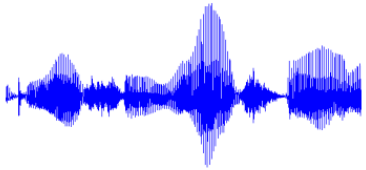
CAP5415-Computer Vision
Lecture 9-Neural Nets for Computer Vision I

Dr. Ulas Bagci

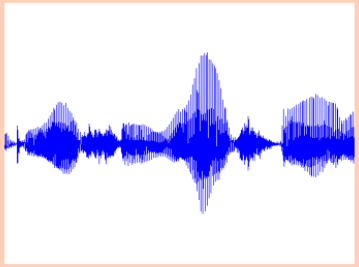
bagci@ucf.edu

Typical goal of machine learning

	input		output
images/video		ML	Label: “Motorcycle” Suggest tags Image search …
audio		ML	Speech recognition Music classification Speaker identification …
text		ML	Web search Anti-spam Machine translation …

Typical goal of machine learning

Feature engineering:
most time consuming!

input			output
images/video		ML	Label: “Motorcycle” Suggest tags Image search …
audio		ML	Speech recognition Music classification Speaker identification …
text		ML	Web search Anti-spam Machine translation …

Our goal in object classification

ML → "motorcycle"

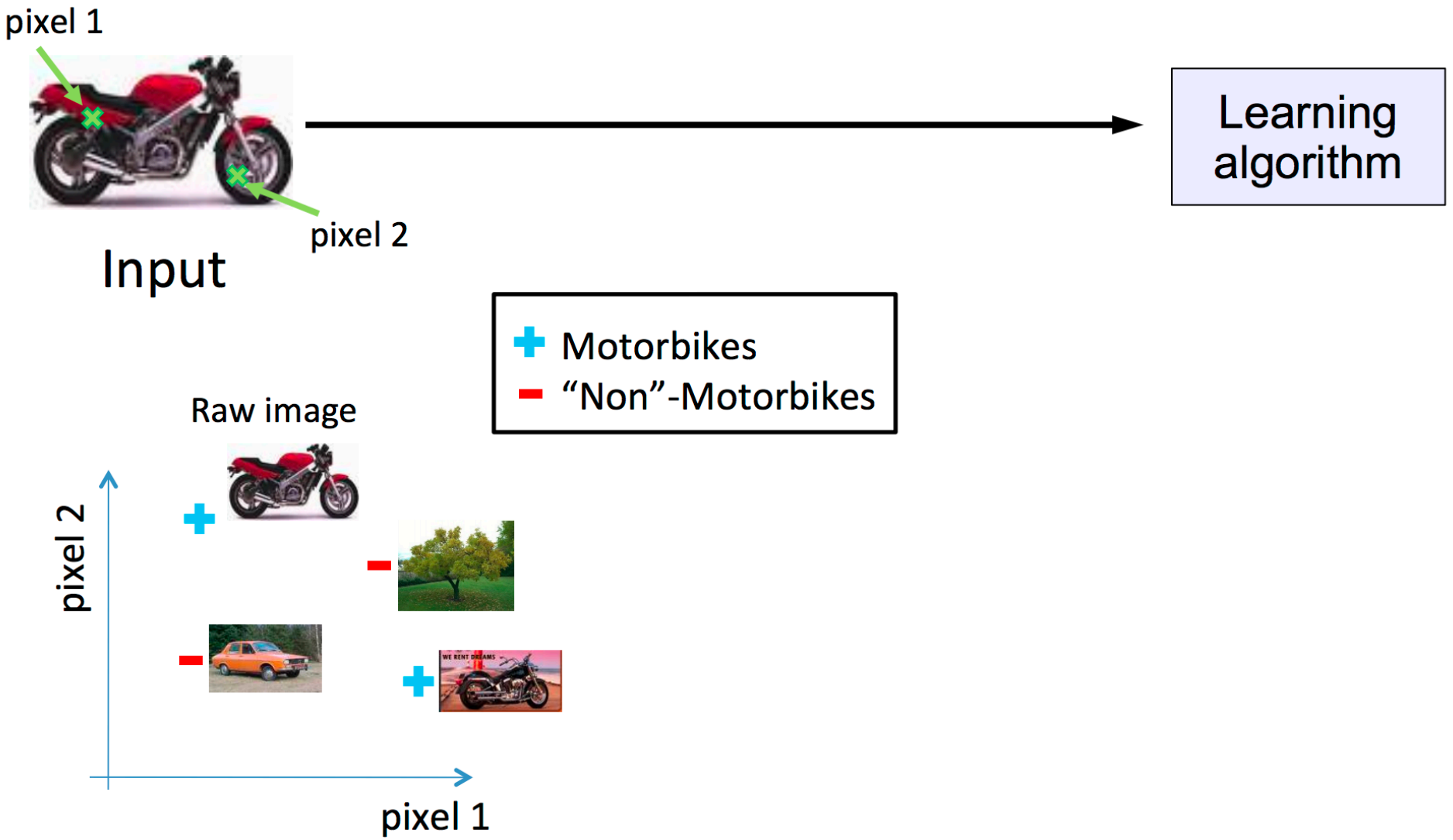
Why is this hard?

You see this:

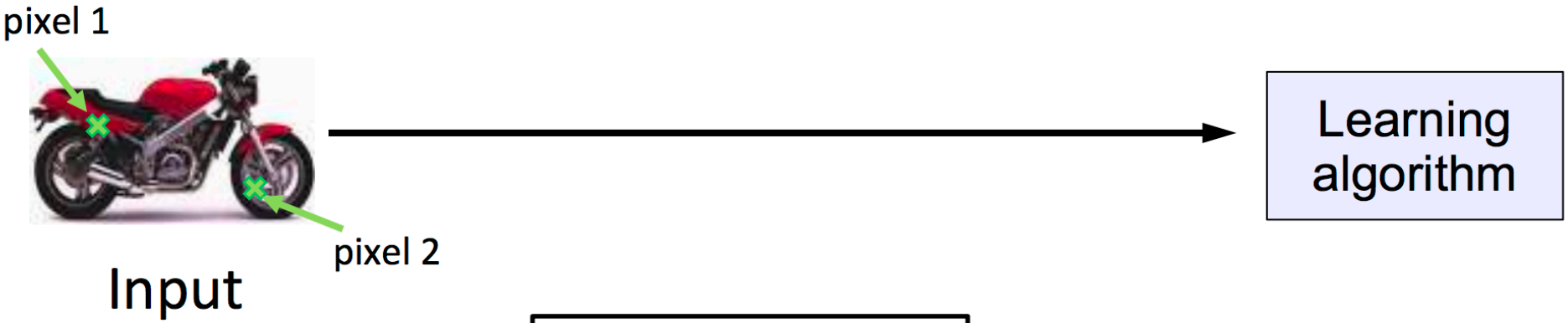
But the camera sees this:

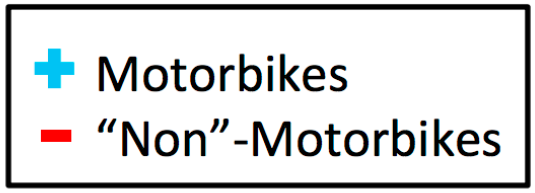
194	210	201	212	199	213	215	195	178	158	182	209
180	189	190	221	209	205	191	167	147	115	129	163
114	126	140	188	176	165	152	140	170	106	78	88
87	103	115	154	143	142	149	153	173	101	57	57
102	112	106	131	122	138	152	147	128	84	58	66
94	95	79	104	105	124	129	113	107	87	69	67
68	71	69	98	89	92	98	95	89	88	76	67
41	56	68	99	63	45	60	82	58	76	75	65
20	43	69	75	56	41	51	73	55	70	63	44
50	50	57	69	75	75	73	74	53	68	59	37
72	59	53	66	84	92	84	74	57	72	63	42
67	61	58	65	75	78	76	73	59	75	69	50

Pixel-based representation

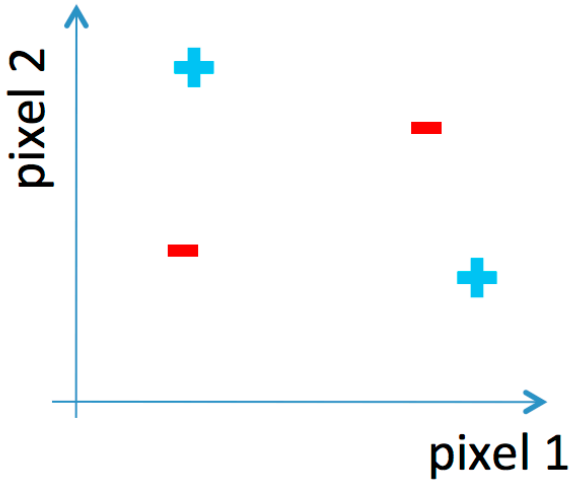


Pixel-based representation

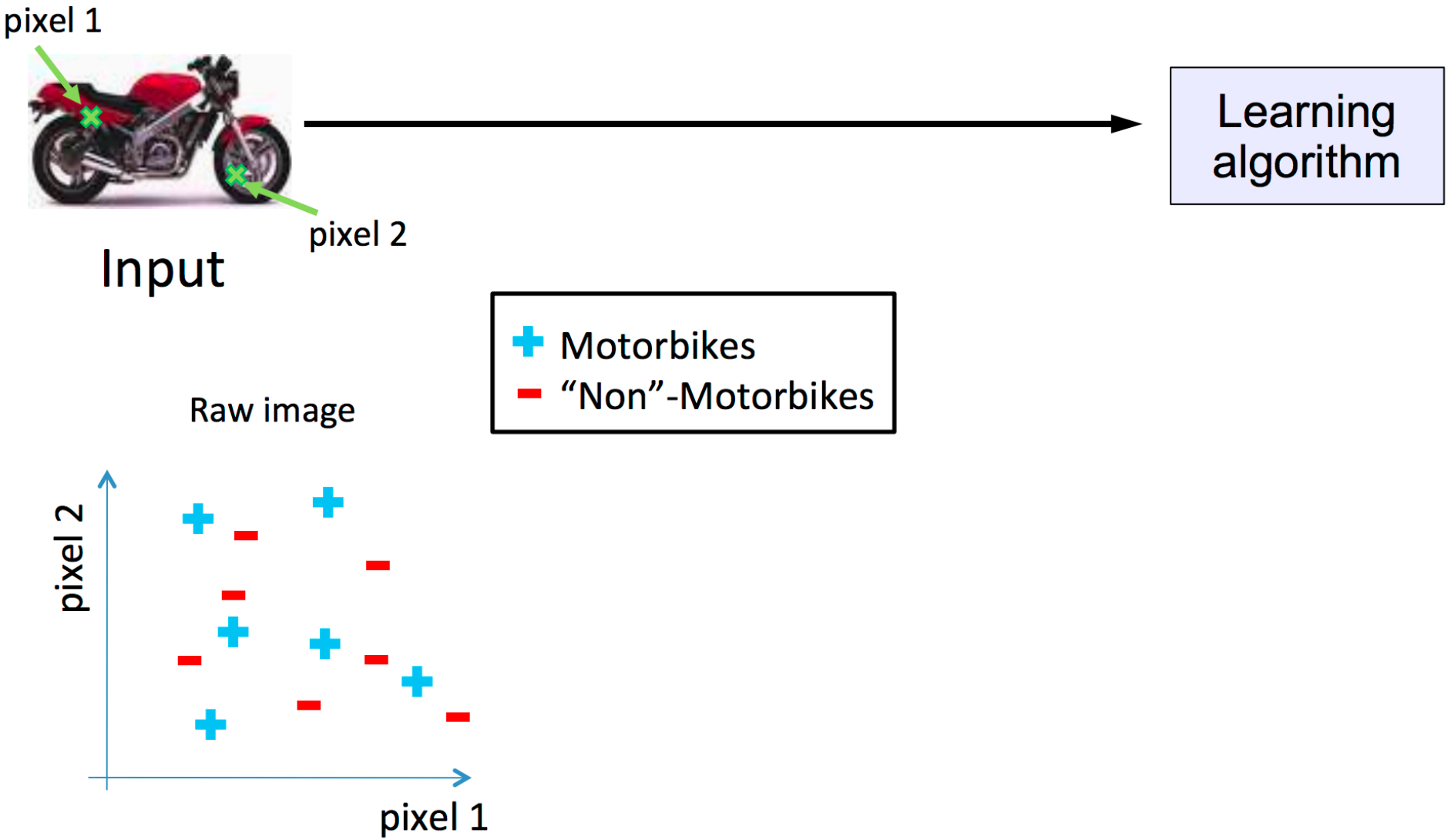




Raw image



Pixel-based representation



What we want

Input

Feature representation

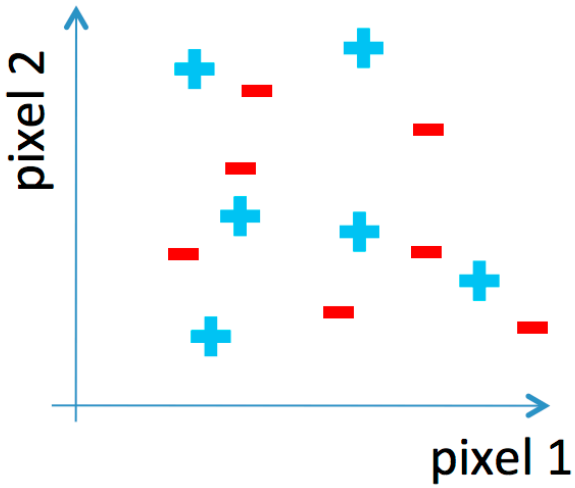
E.g., Does it have Handlebars? Wheels?

Learning algorithm

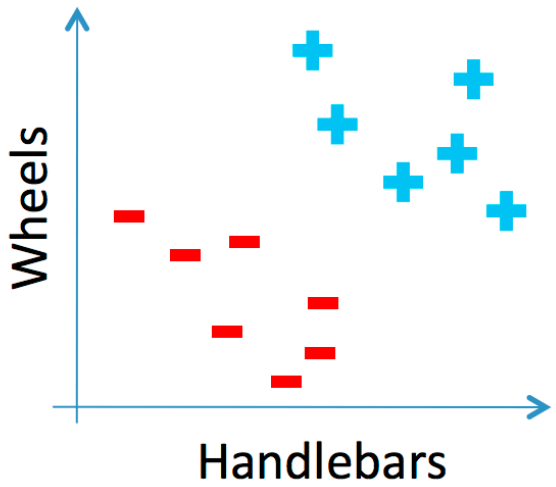
Motorbikes

"Non"-Motorbikes

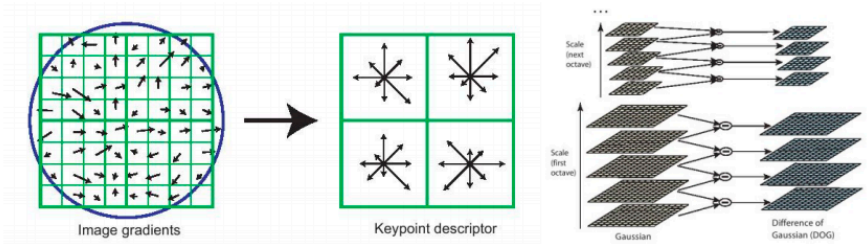
Raw image



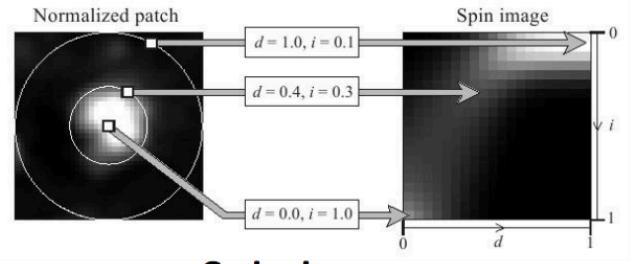
Features



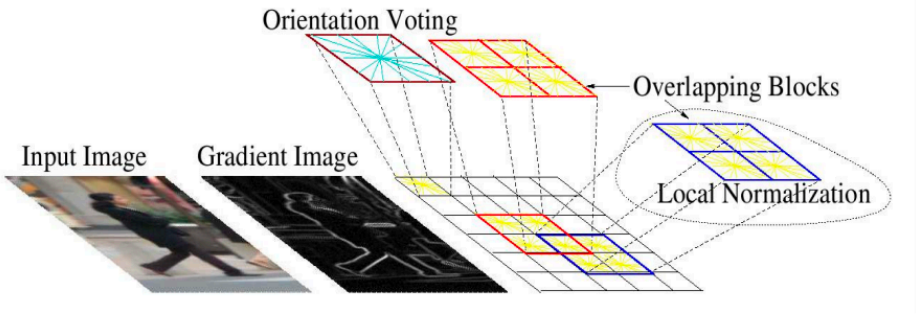
Some feature representations



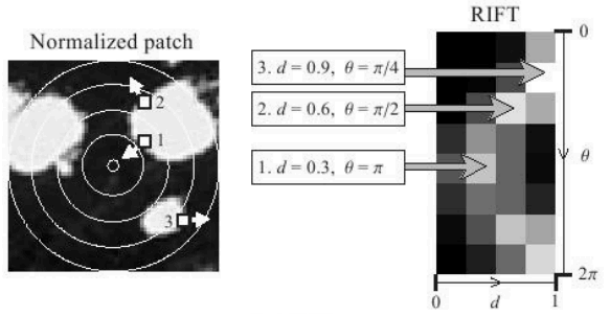
SIFT



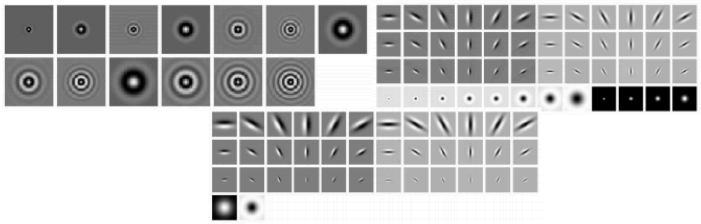
Spin image



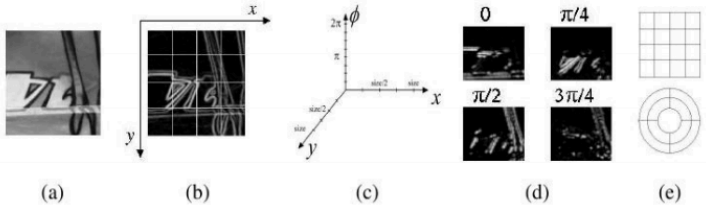
HoG



RIFT

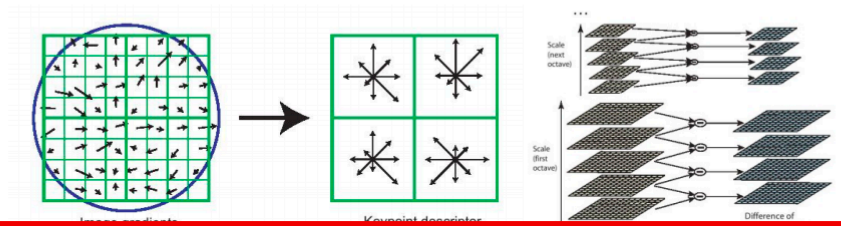


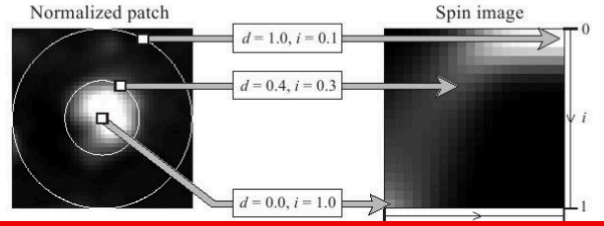
Textons



GLOH

Some feature representations

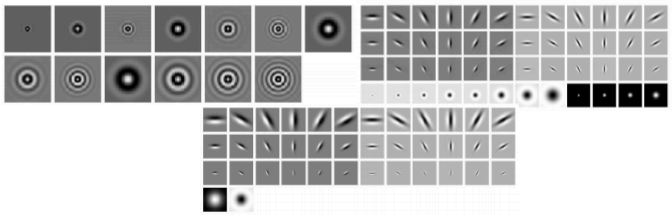




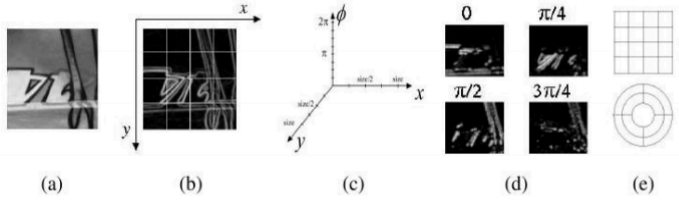
Coming up with features is often difficult, time-consuming, and requires expert knowledge.

HoG

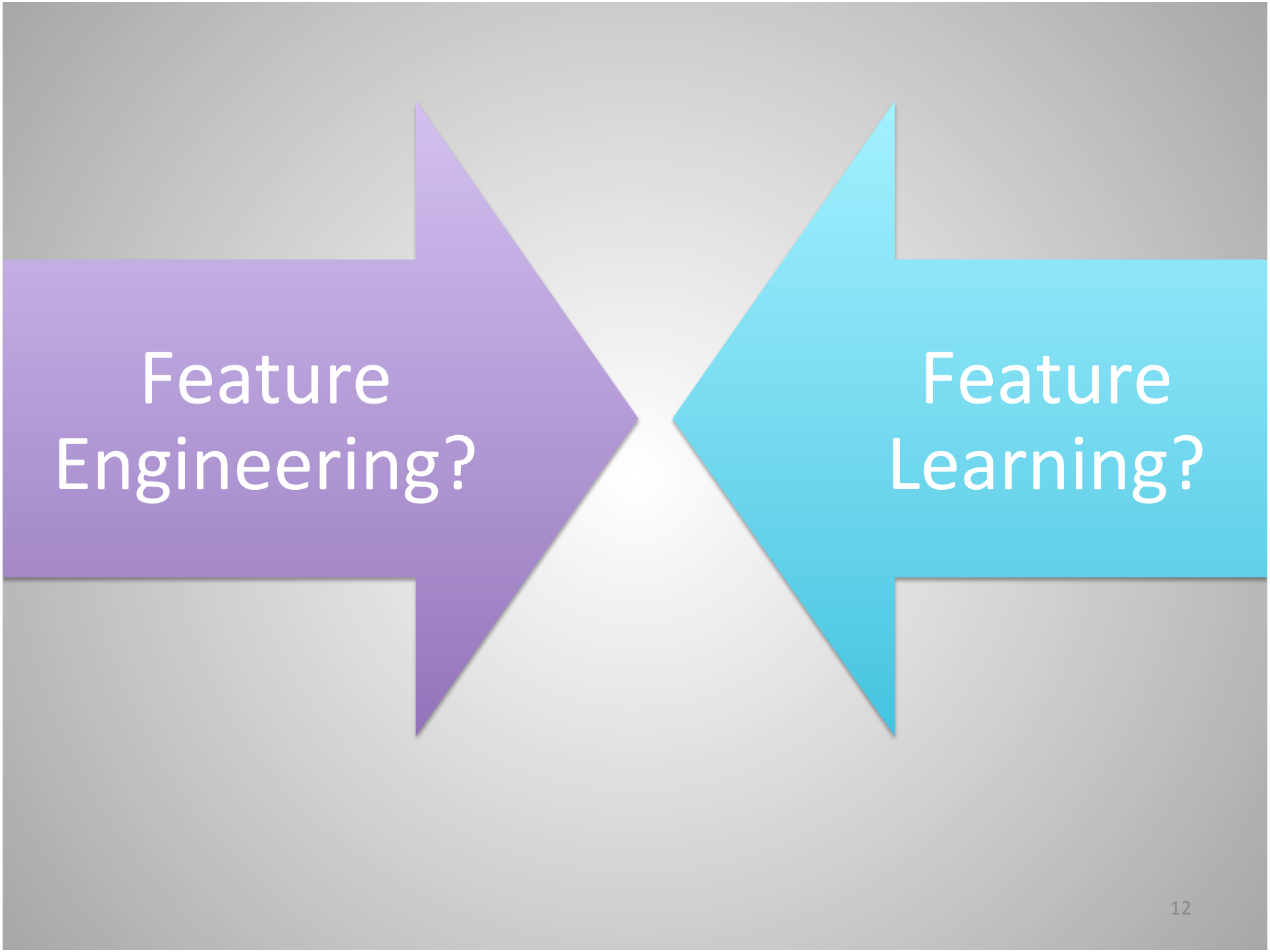
RIFT



Textons



GLOH


Feature Engineering?
Feature Learning?

Representation Learning

- Is there some way to extract **meaningful features** from data in a *supervised* or *unsupervised* manner?

Representation Learning

- Is there some way to extract **meaningful features** from data in a *supervised* or *unsupervised* manner?

- Biologically inspired systems
  - to make the computer more robust, intelligent, and learn, …

Representation Learning

- Is there some way to extract **meaningful features** from data in a *supervised* or *unsupervised* manner?

- Biologically inspired systems
  - to make the computer more robust, intelligent, and learn, …
  - Model our systems after **the brain**!

Representation Learning

- Is there some way to extract **meaningful features** from data in a *supervised* or *unsupervised* manner?

- Biologically inspired systems
  - to make the computer more robust, intelligent, and learn, …
  - Model our systems after **the brain**!
    - Brain interprets imprecise information from the senses at an incredibly rapid rate!
    - It discerns a whisper in a noisy room, a face in a dimly lit alley, and hidden agenda in a political statement.

A common logic to seeing cats and cosmos

There may be a universal logic to how physicists, computers and brains tease out important features from among other irrelevant bits of data.

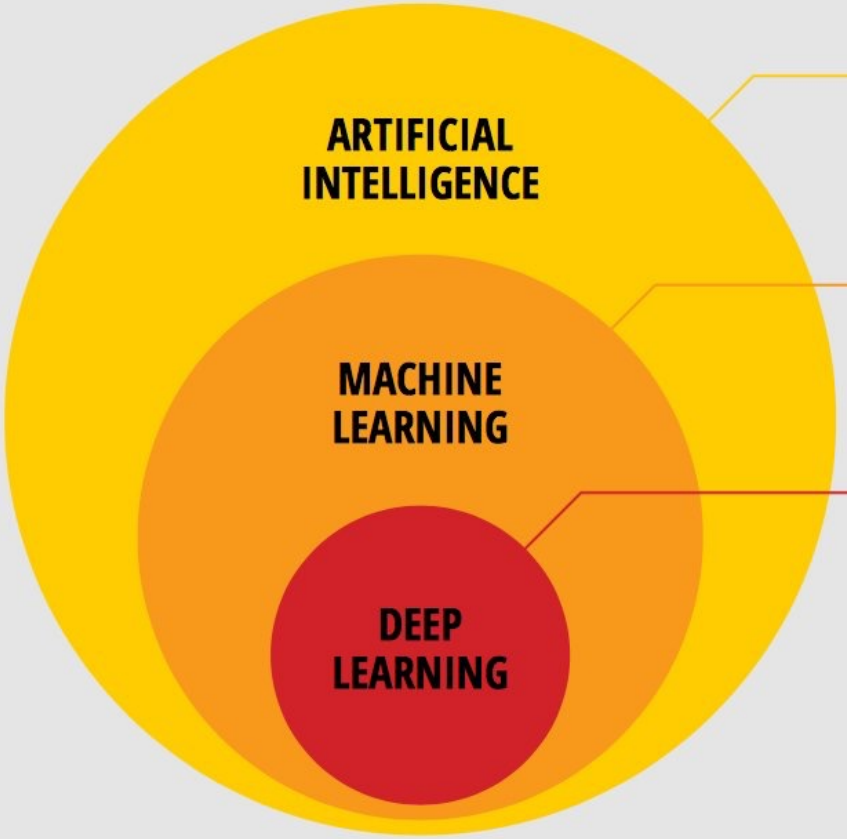
Quanta, 2014

Types of Learning

- **Supervised (inductive) learning**
  - Training data includes desired outputs
- **Unsupervised learning**
  - Training data does not include desired outputs
- **Semi-supervised learning**
  - Training data includes a few desired outputs
- **Reinforcement learning**
  - Rewards from sequence of actions

AI, ML, and DL

Figure 1 — Artificial intelligence (AI), machine learning and deep learning[7]



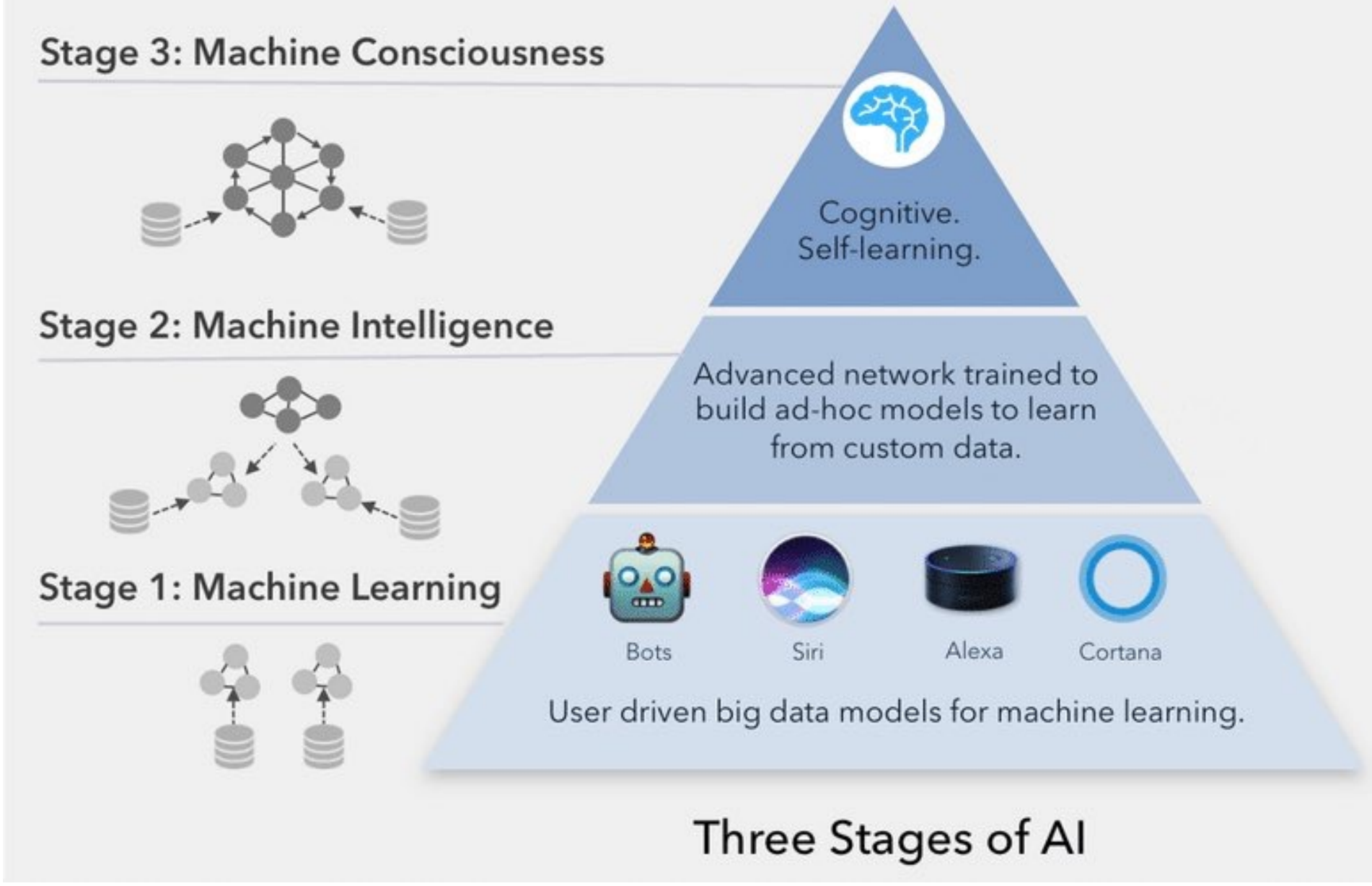
ARTIFICIAL INTELLIGENCE is the study of devices that perceive their environment and define a course of action that will maximize its chance of achieving a given goal.[8]

MACHINE LEARNING is a subset of artificial intelligence, in which machines learn how to to complete a certain task without being explicitly programmed to do so.

DEEP LEARNING is a subset of machine learning in which the tasks are broken down and distributed onto machine learning algorithms that are organised in consecutive layers. Each layer builds up on the output from the previous layer. Together the layers constitute an artificial neural network that mimics the distributed approach to problem-solving carried out by neurons in a human brain.

source www.webfoundation.org via mikequindazzi

Where does AI go?



Use of Neural Networks (…& deep NN)

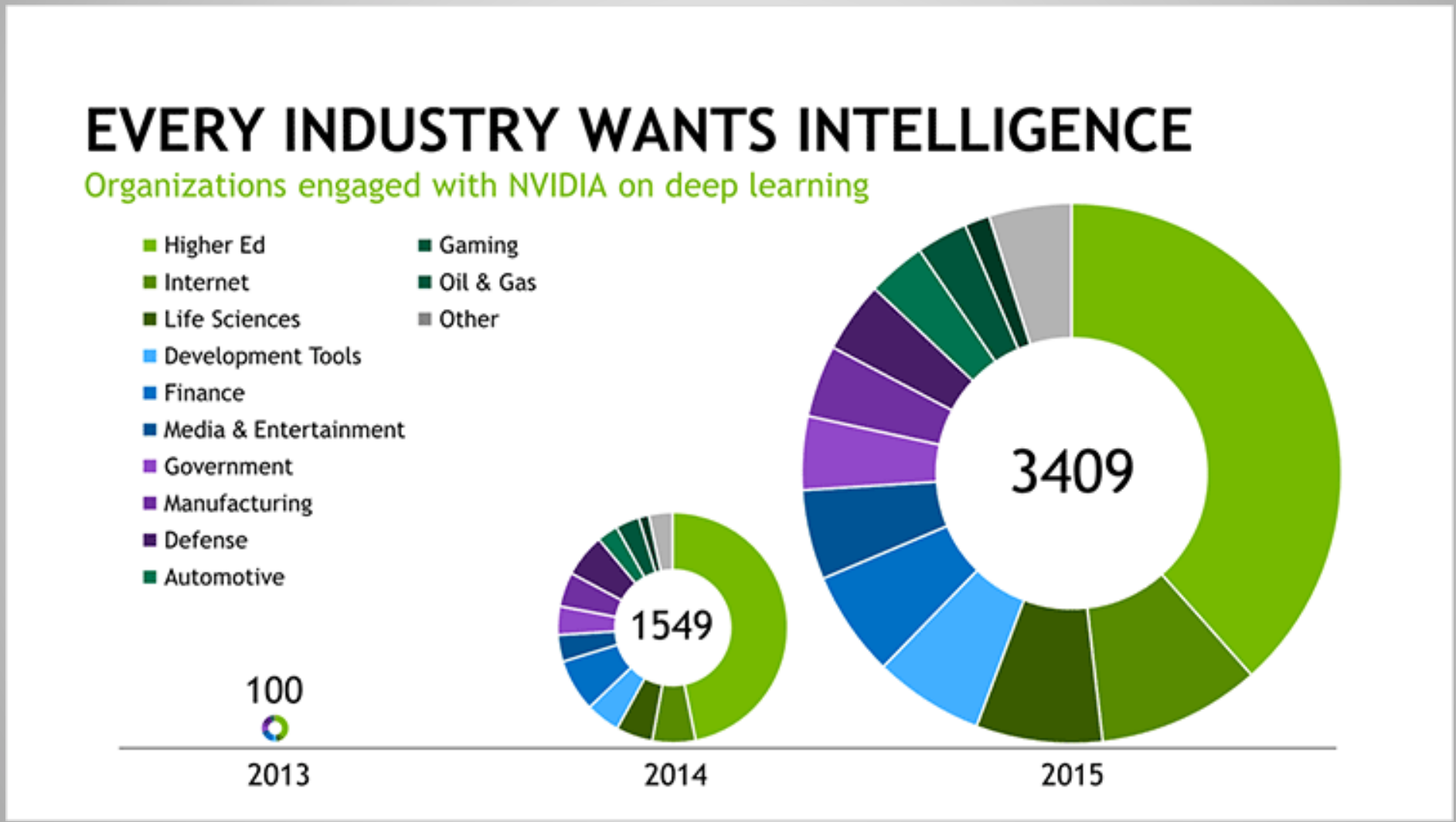
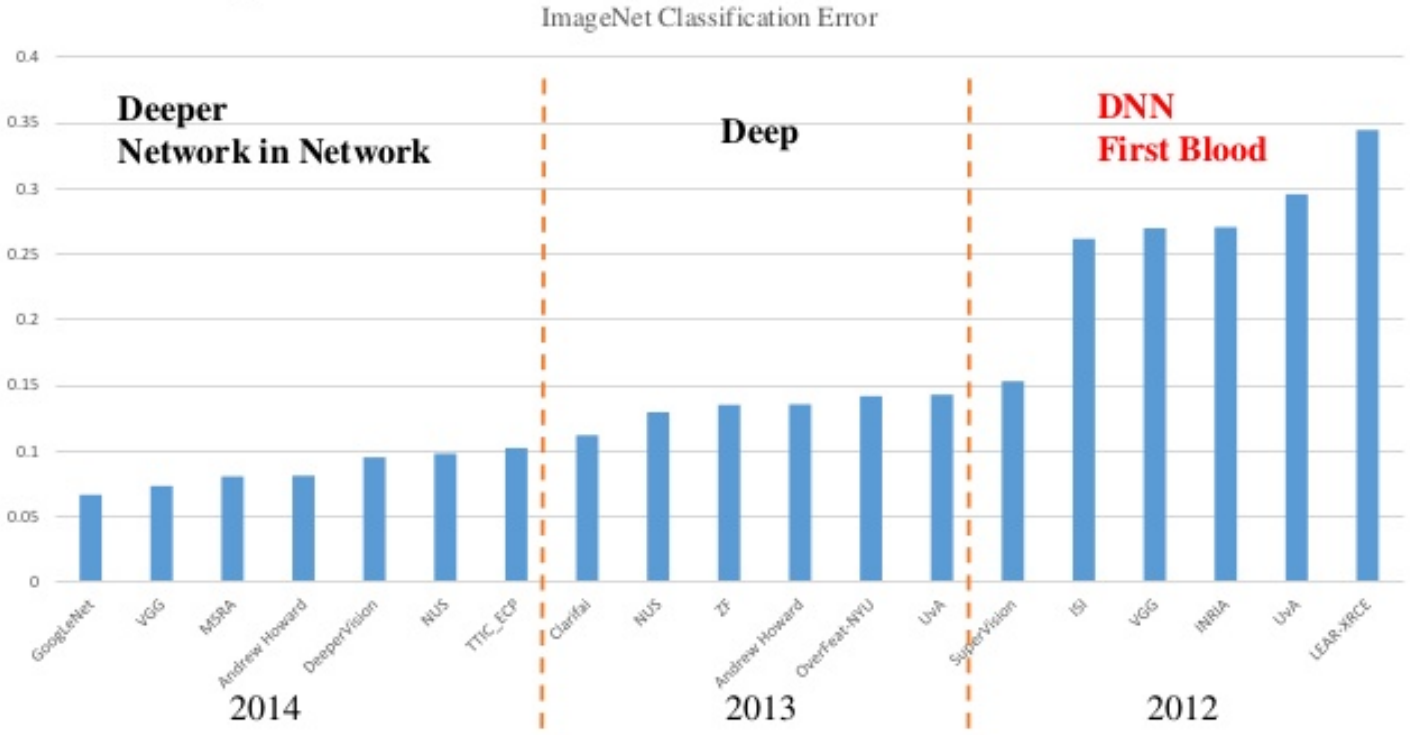


Image Classification

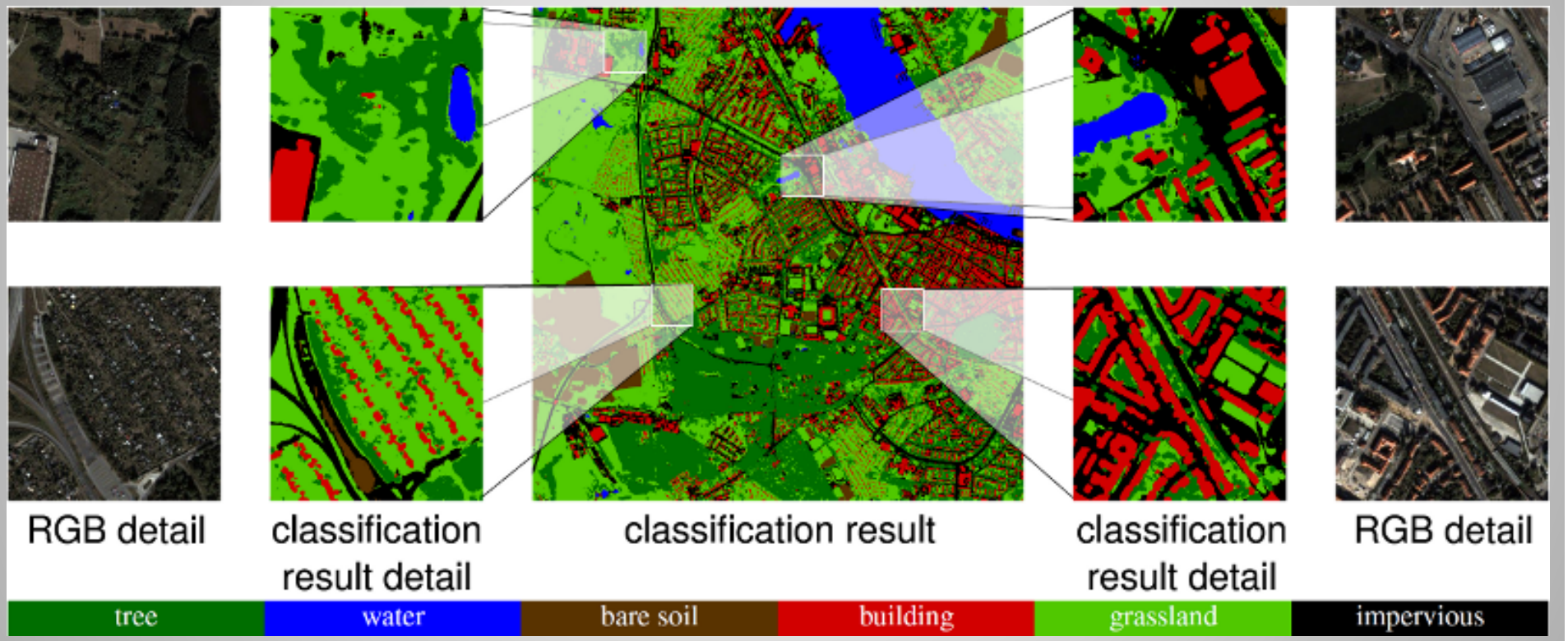
ImageNet Classification

- **1000** categories and **1.2** million training images

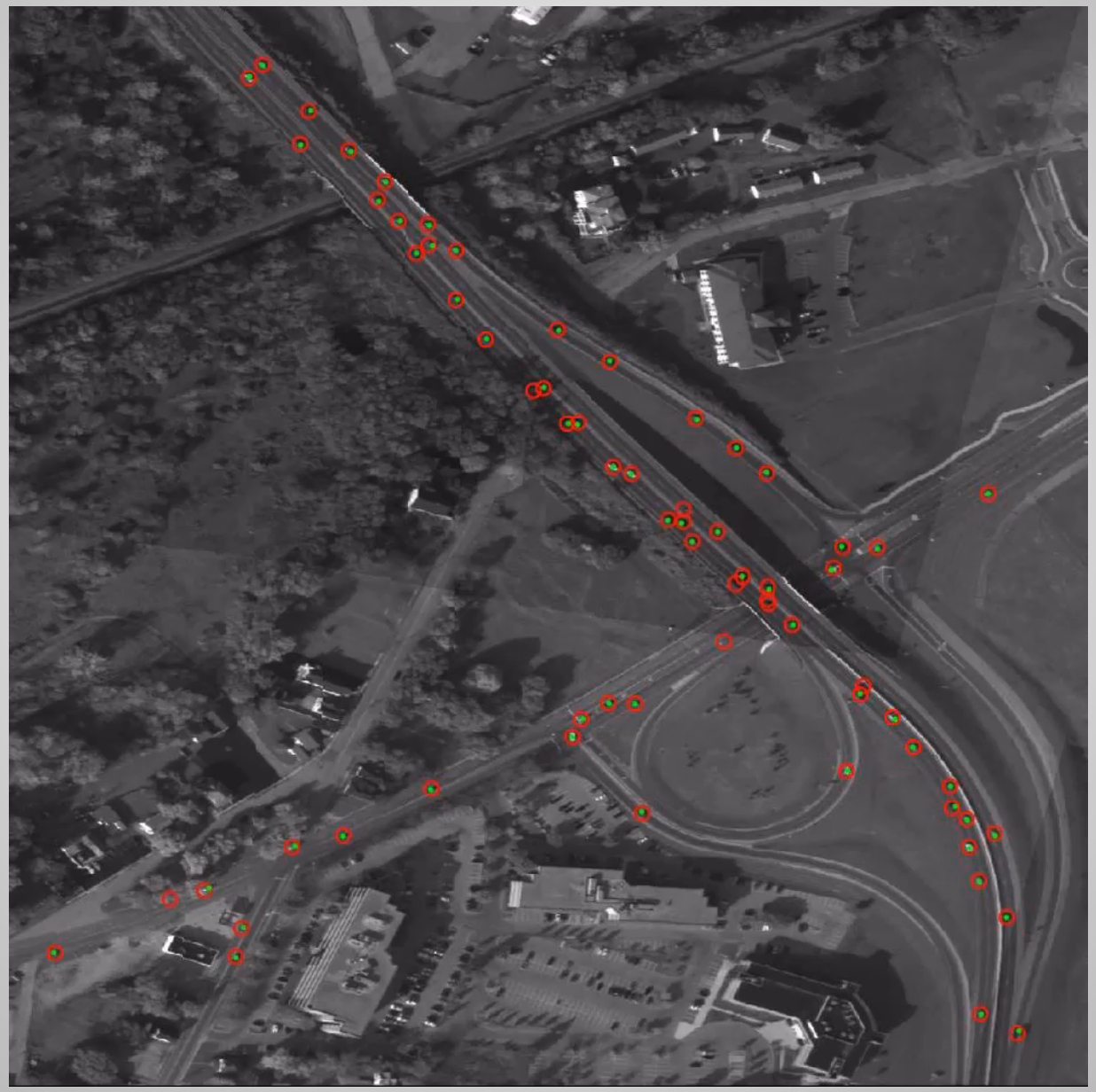


Li Fei-Fei: ImageNet Large Scale Visual Recognition Challenge, 2014 http://image-net.org/

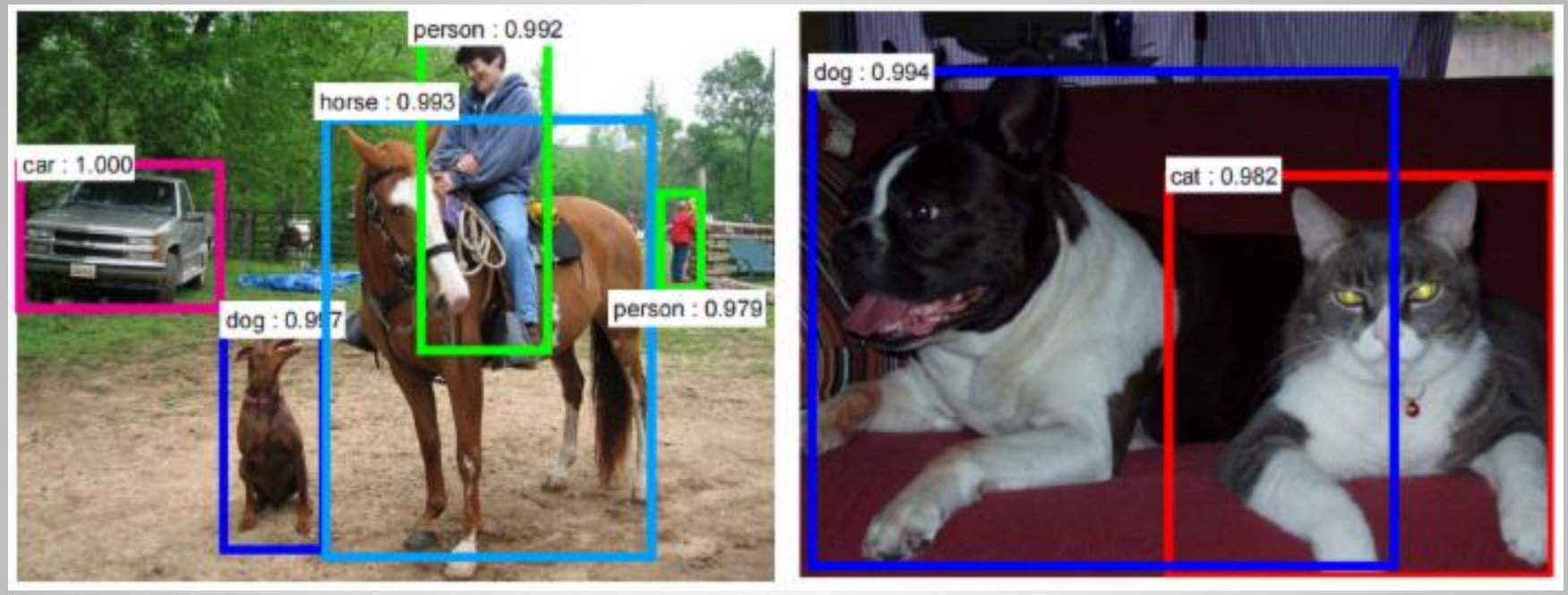
Semantic Segmentation

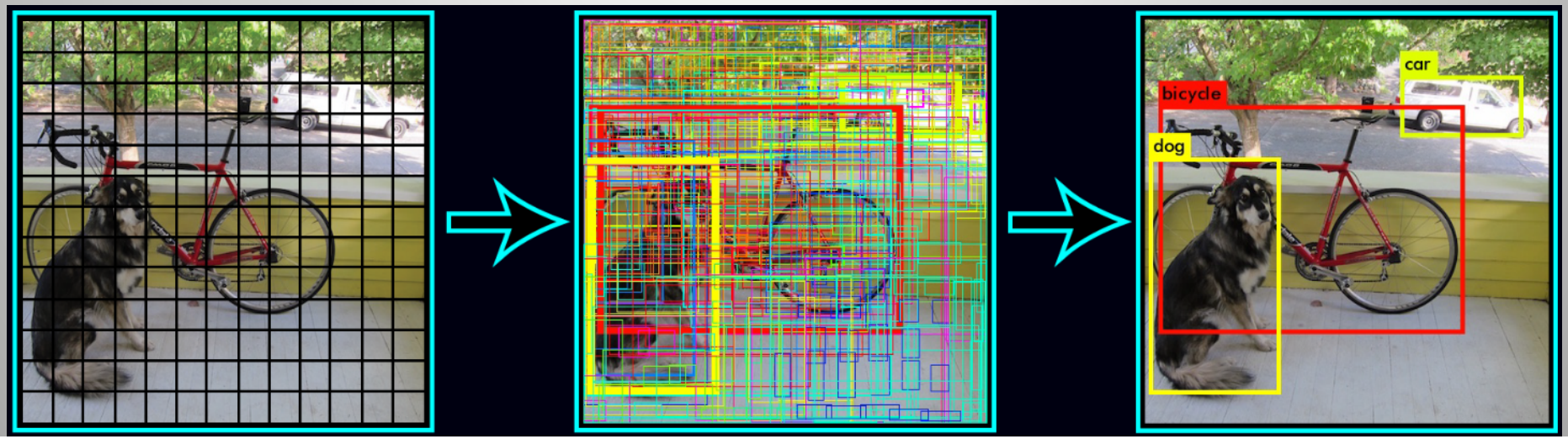


Object Detection/Tracking

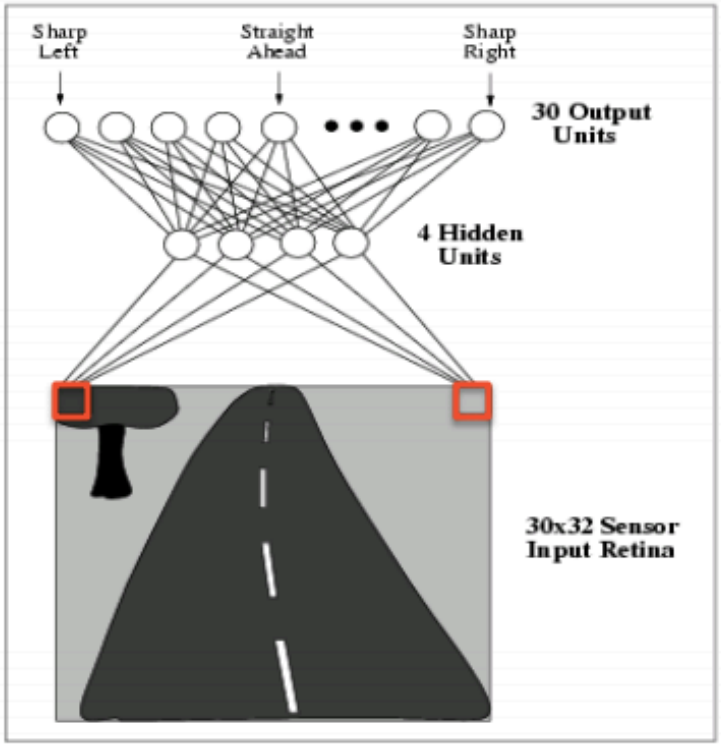


Object Detection

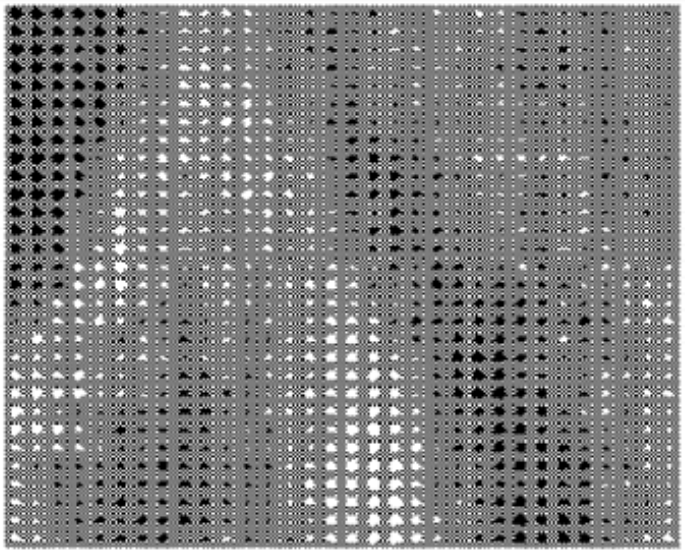




Neural Network trained to drive a car!

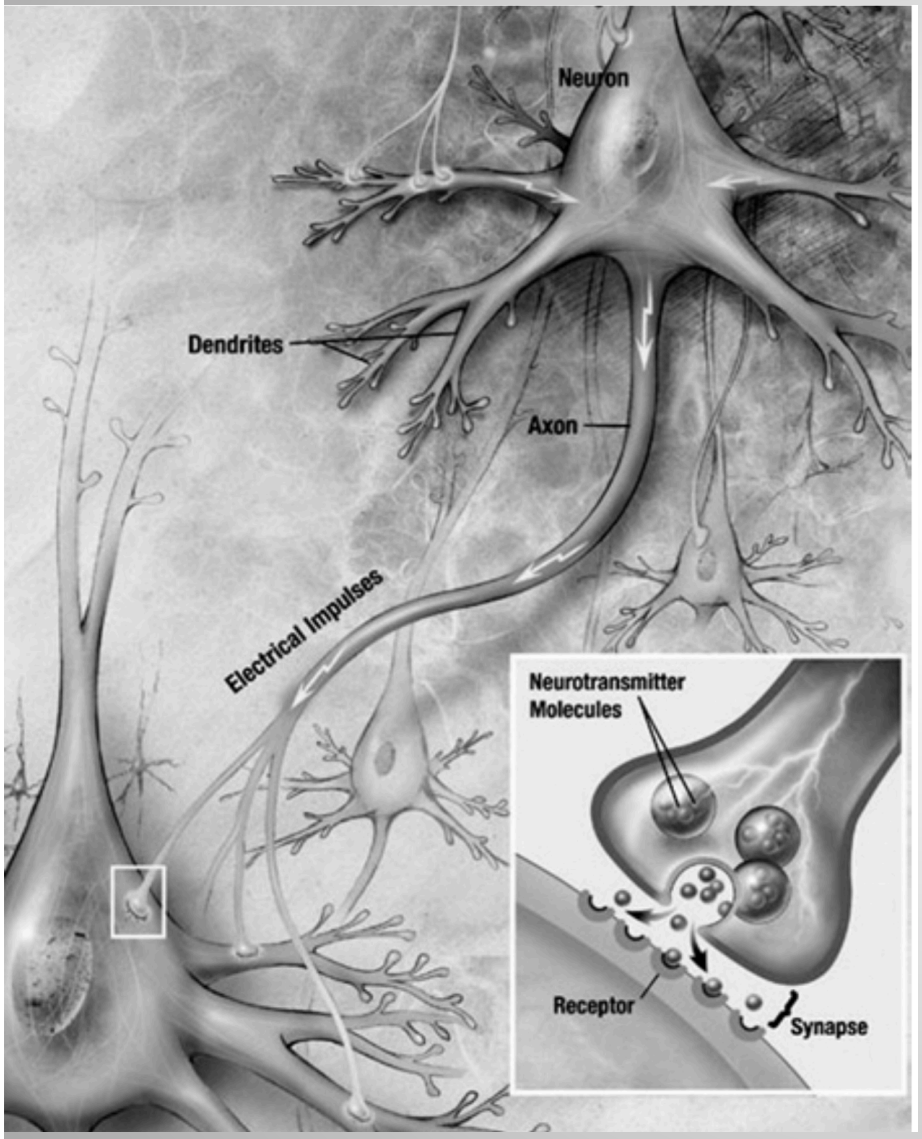


Weights to output units from the hidden unit

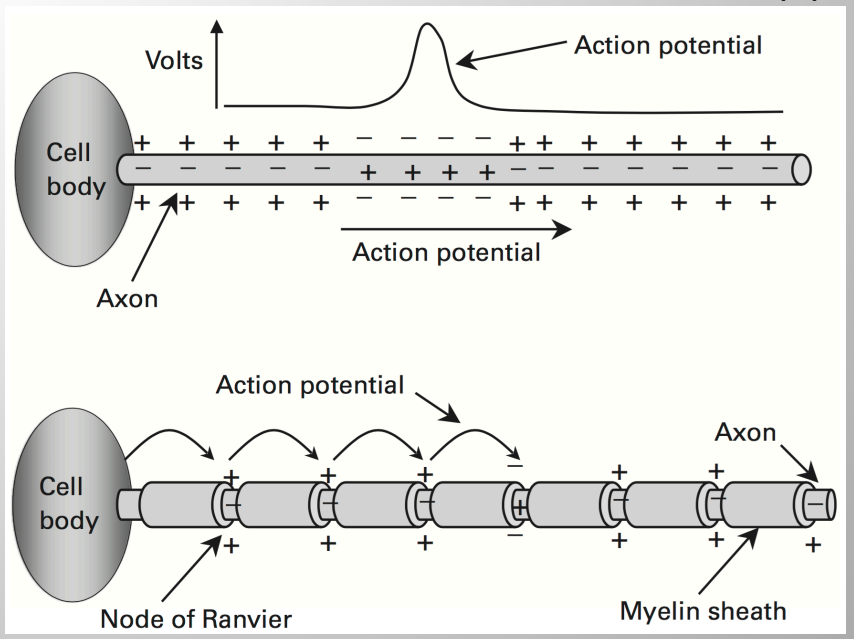


Weights of each pixel for one hidden unit

Neurons in the Brain

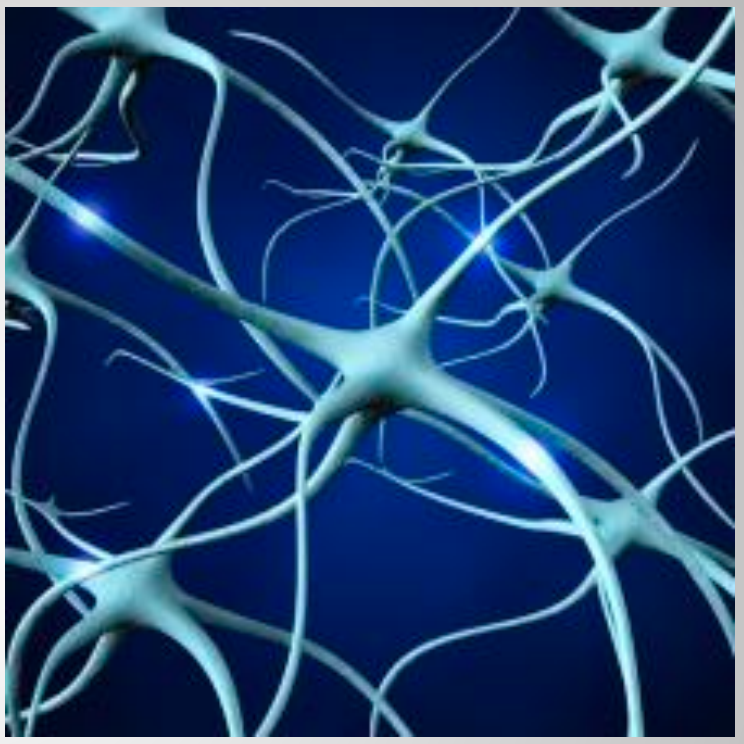


- Brain is composed of **neurons**
- A neuron receives input from other neurons (generally thousands) from its synapses
- Inputs are approximately **summed**
- When the input exceeds a threshold the neuron sends an electrical spike that travels that travels from the body, down the axon, to the next neuron(s)



Background in Neural Nets (NN)

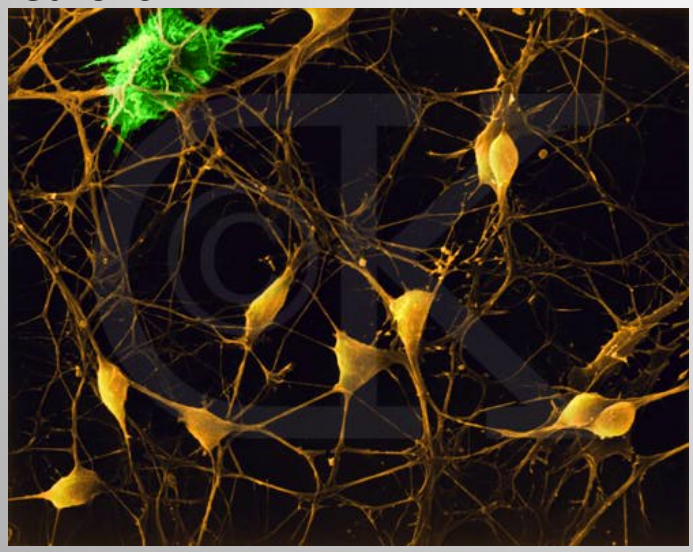
- Neural Nets can be:
  - **Biological** Models
  - **Artificial** Models



- Desire to produce artificial systems capable of sophisticated computations similar to human brain!

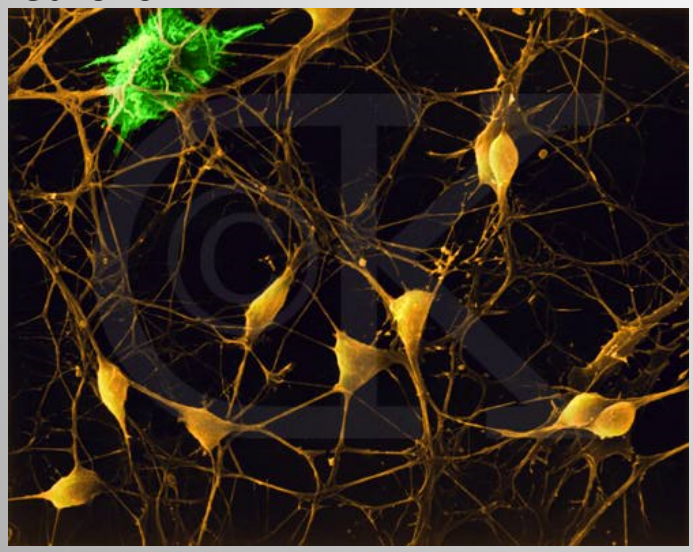
Brain Analogy of NN: Brain is a remarkable Computer

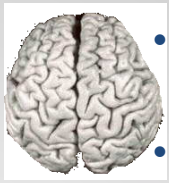
- The brain is composed of a mass of interconnected neurons
  - each neuron is connected to many other neurons
- Neurons transmit signals to each other
- Whether a signal is transmitted is an all-or-nothing event (the electrical potential in the cell body of the neuron is thresholded)
- Whether a signal is sent, depends on the strength of the bond (synapse) between two neurons



Brain Analogy of NN: Brain is a remarkable Computer

- The brain is composed of a mass of interconnected neurons
  - each neuron is connected to many other neurons
- Neurons transmit signals to each other
- Whether a signal is transmitted is an all-or-nothing event (the electrical potential in the cell body of the neuron is thresholded)
- Whether a signal is sent, depends on the strength of the bond (synapse) between two neurons

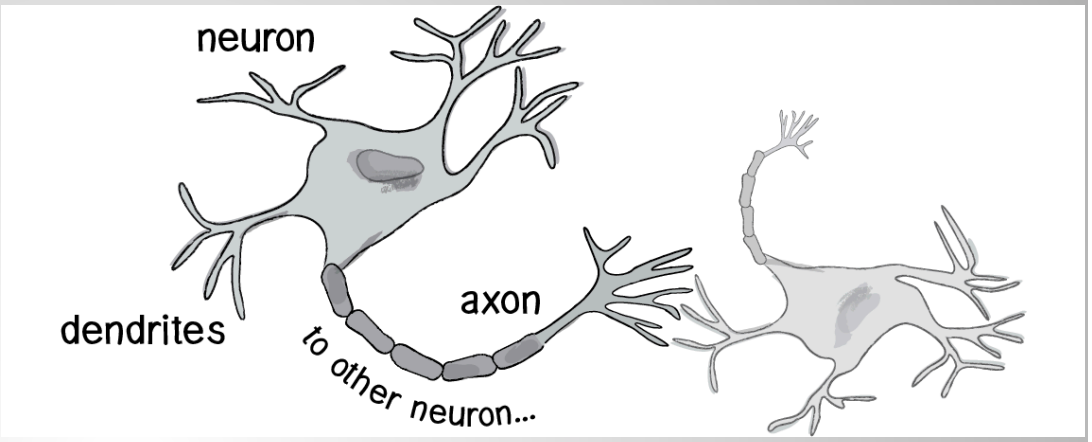


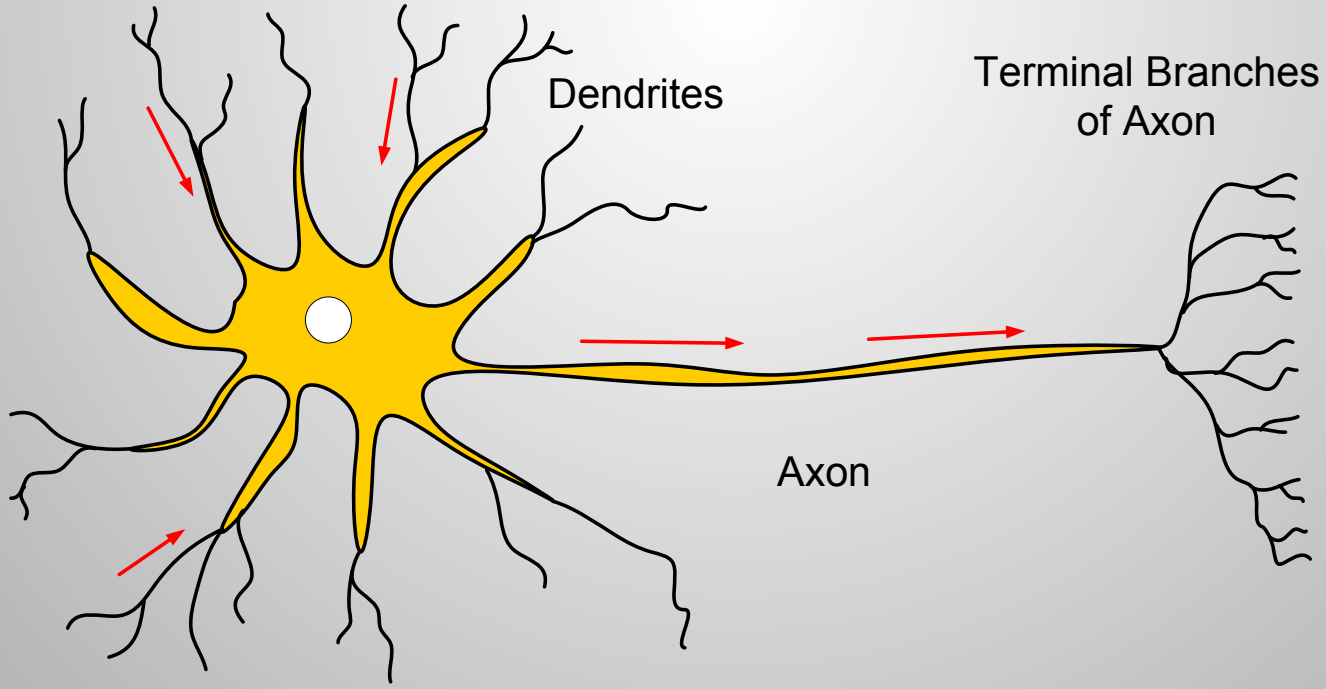


- 200 billion neurons, 32 trillion synapses
- Element size: 10^{-6} m
- Energy use: 25W
- Processing speed: 100 Hz
- Parallel, Distributed
- Fault Tolerant
- Learns: Yes
- Intelligent/Conscious: Usually

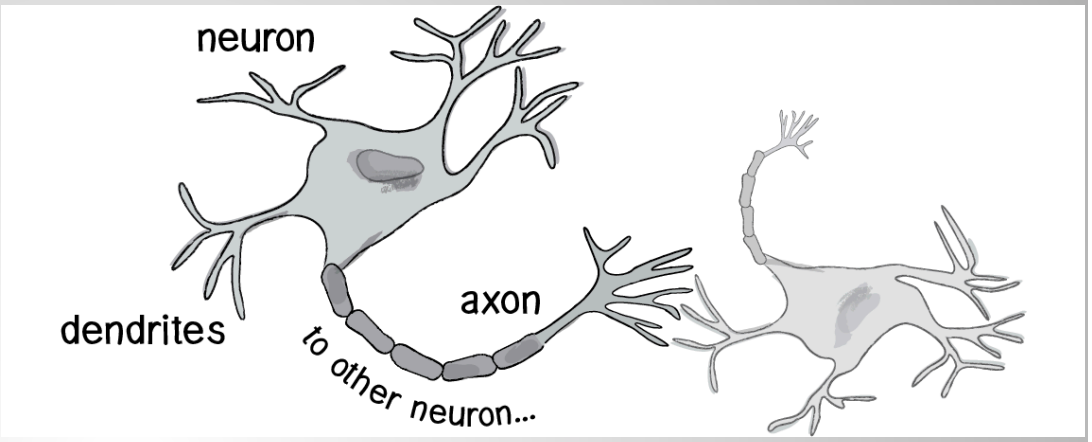
- 1 billion bytes RAM but trillions of bytes on disk
- Element size: 10^{-9} m
- Energy watt: 30-90W (CPU)
- Processing speed: 10^{9} Hz
- Serial, Centralized
- Generally not Fault Tolerant
- Learns: Some
- Intelligent/Conscious: Generally No

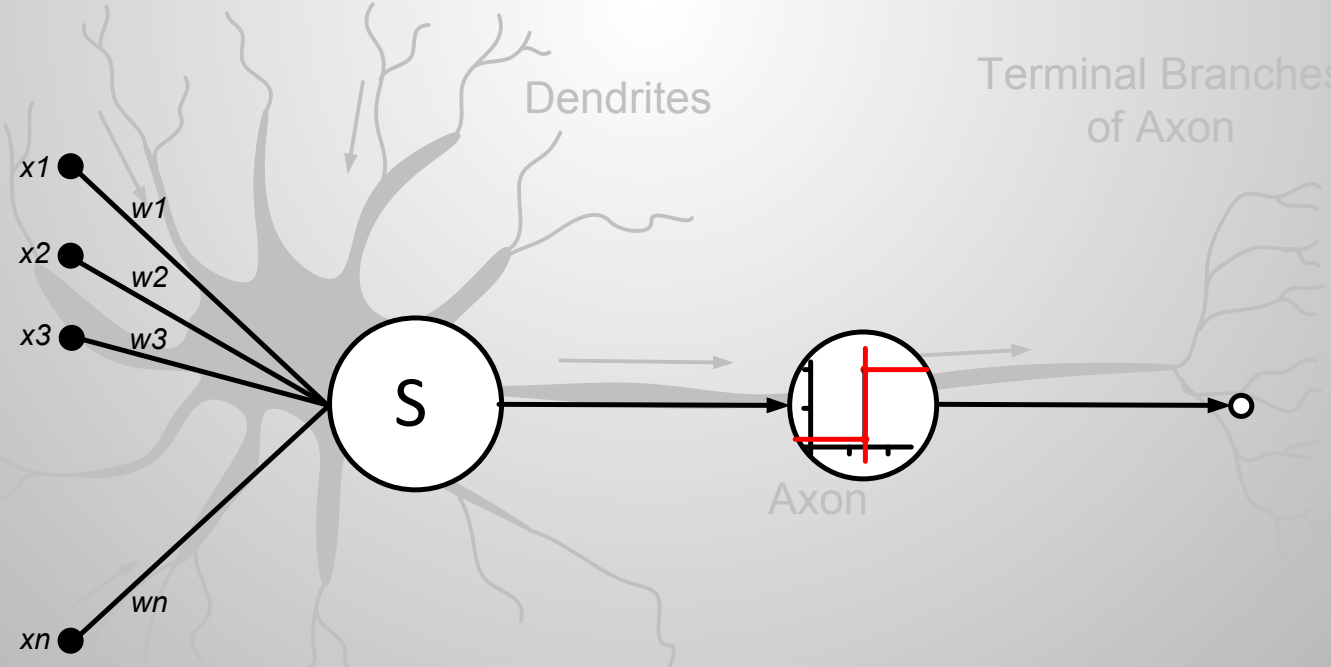
Computational Implementation of the Neural Activation Function



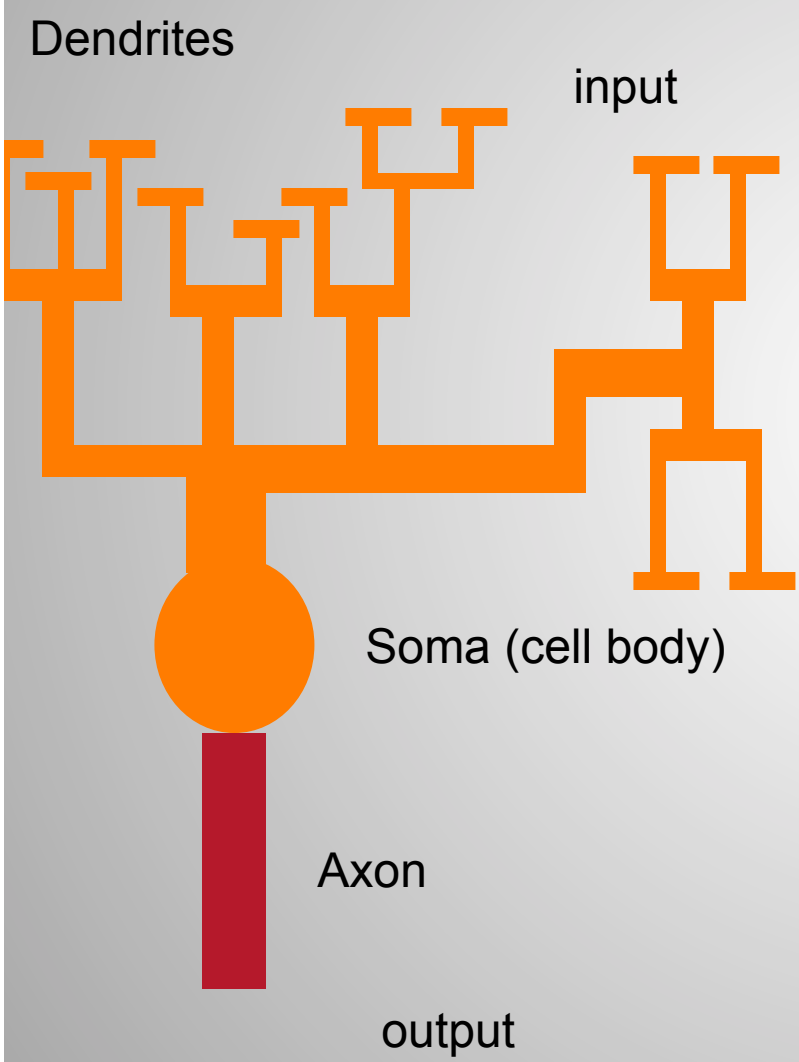


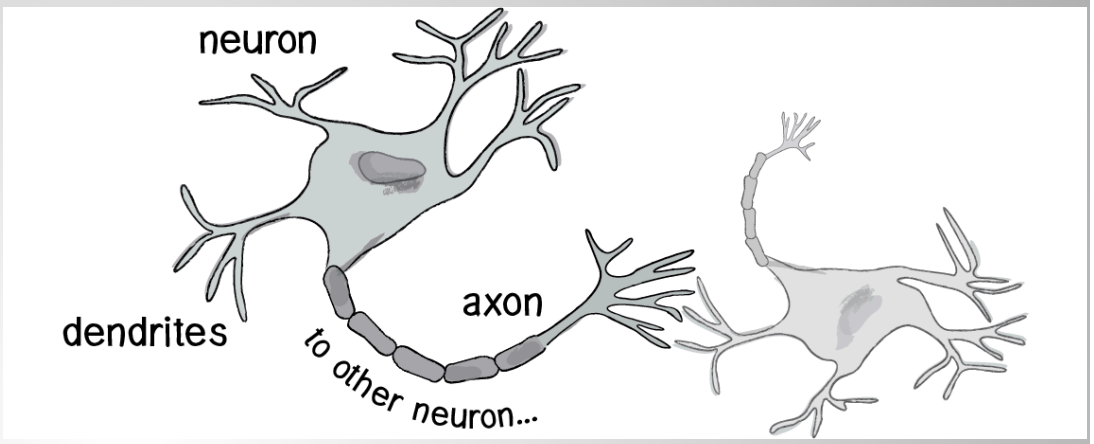
Computational Implementation of the Neural Activation Function



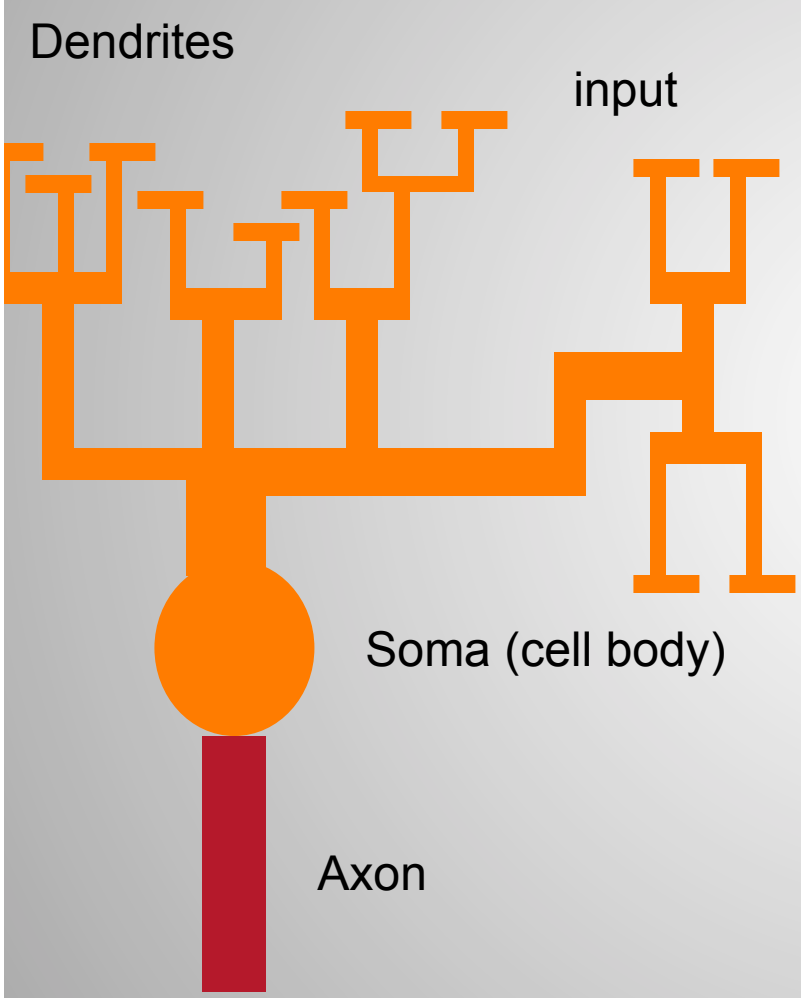


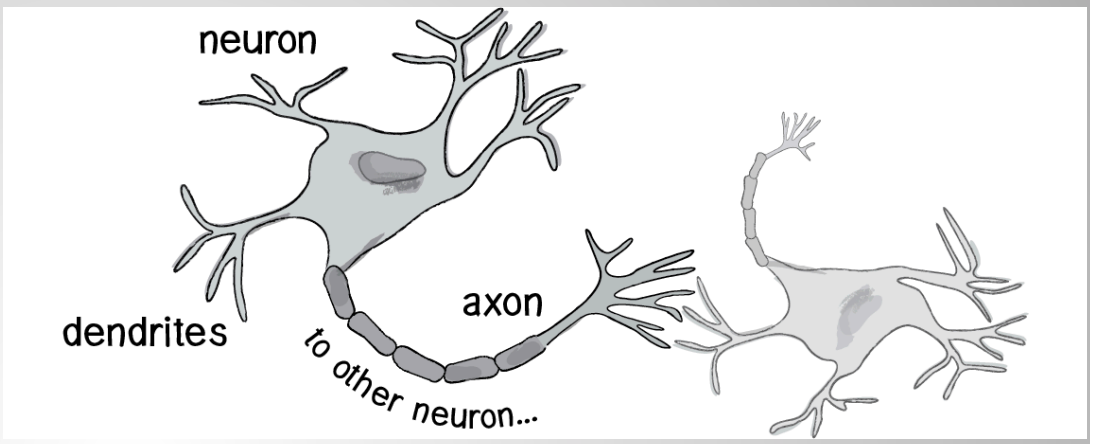
Computational Implementation of the Neural Activation Function





Computational Implementation of the Neural Activation Function





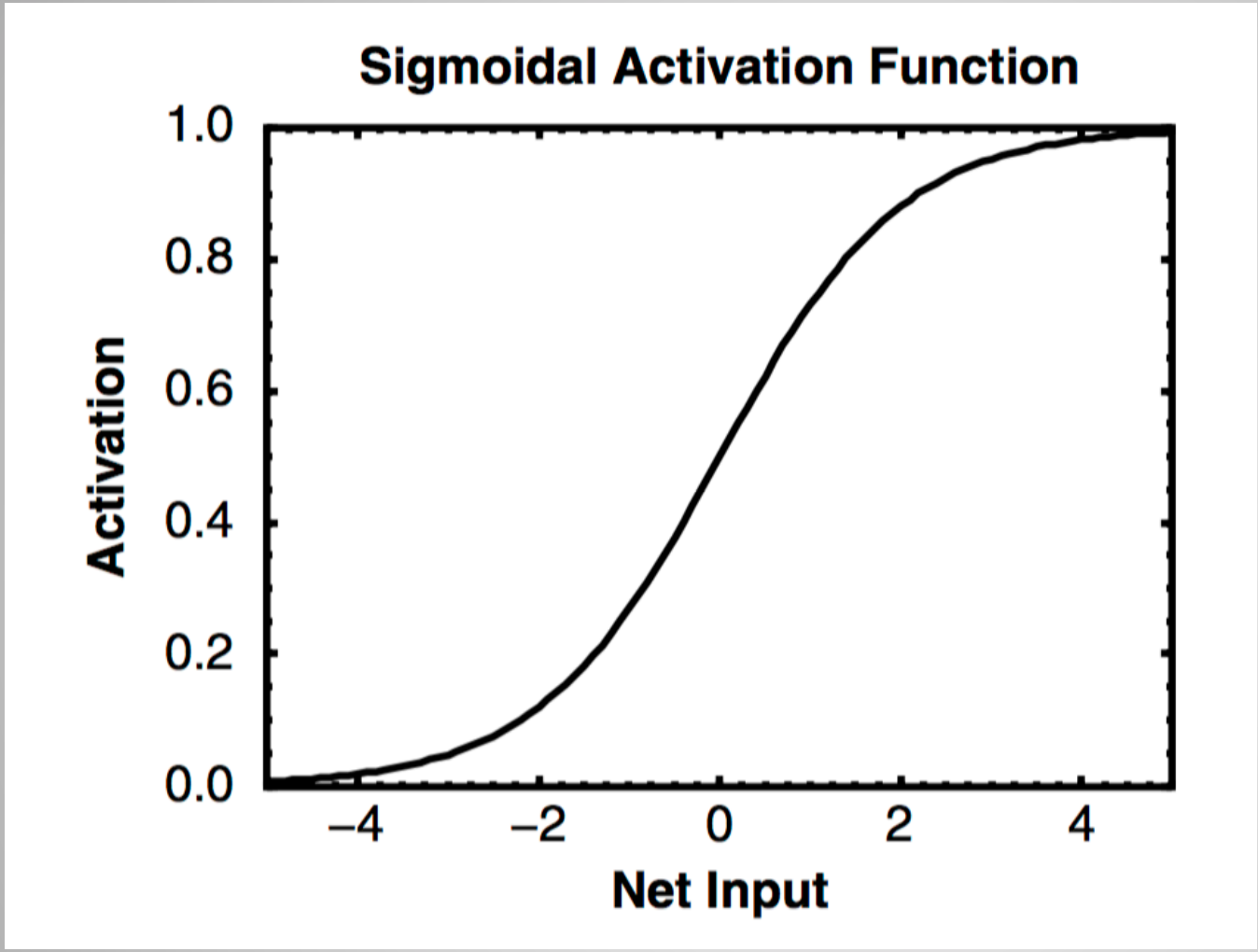
Simple activation function (sum of individual inputs):

$$n_j = \sum_i x_i w_{ij}$$

Activation value (sigmoid / logistic):

$$y_j = \frac{1}{1 + e^{-n_j}}$$

Why Sigmoid?



Step function
Sign function
Sigmoid function

- It is a standard function in NN.
- It transforms the net input value into an activation value y, which is Then sent on to other units

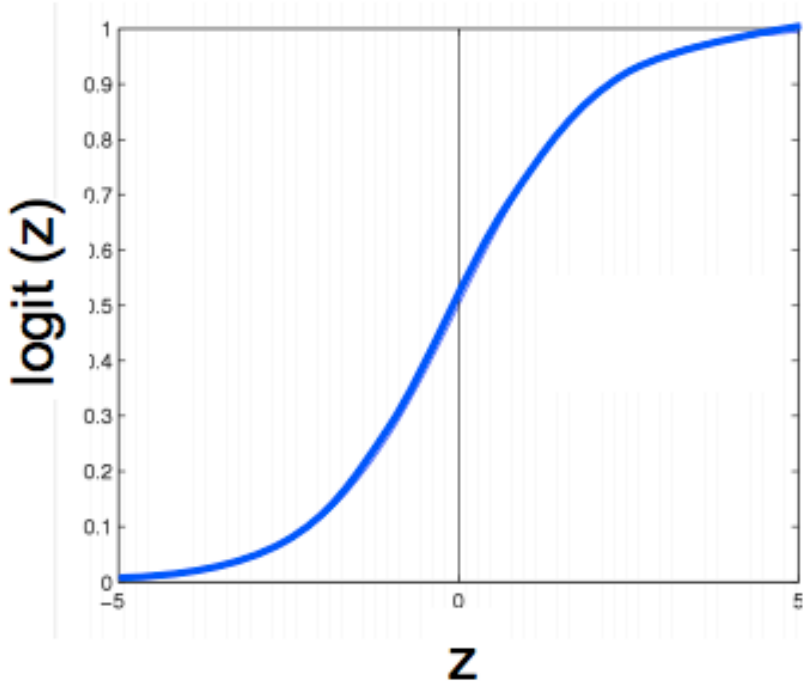
Background on Logit

Assumes the following functional form for P(Y|X):

$$P(Y=1|X) = \frac{1}{1+\exp(-(w_0 + \sum_i w_i X_i))}$$

Logistic function applied to a linear function of the data

Logistic function (or Sigmoid): $\frac{1}{1+exp(-z)}$



Background on Logit

Logistic Regression is a Linear Classifier!

Assumes the following functional form for P(Y|X):

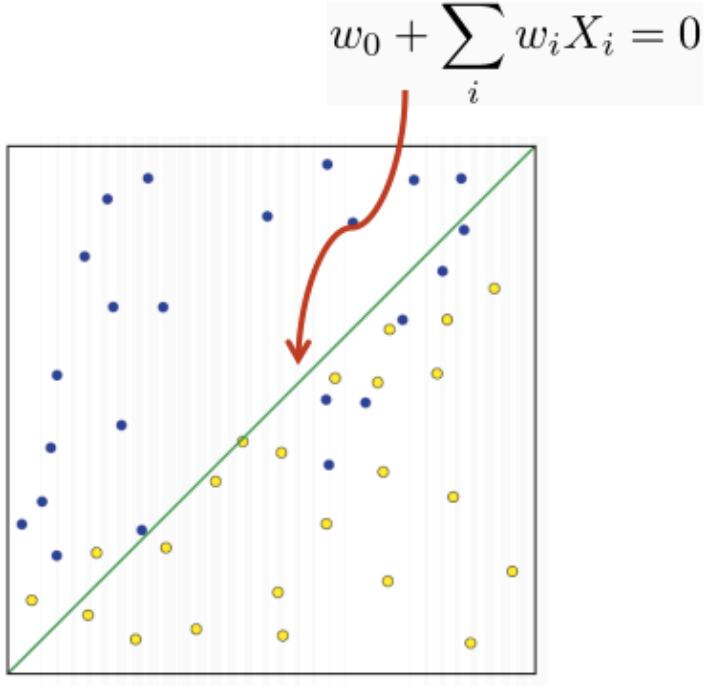
$$P(Y=1|X) = \frac{1}{1+\exp(-(w_0+\sum_i w_i X_i))}$$

Decision boundary:

$$P(Y=0|X) \underset{1}{\overset{0}{\gtrless}} P(Y=1|X)$$

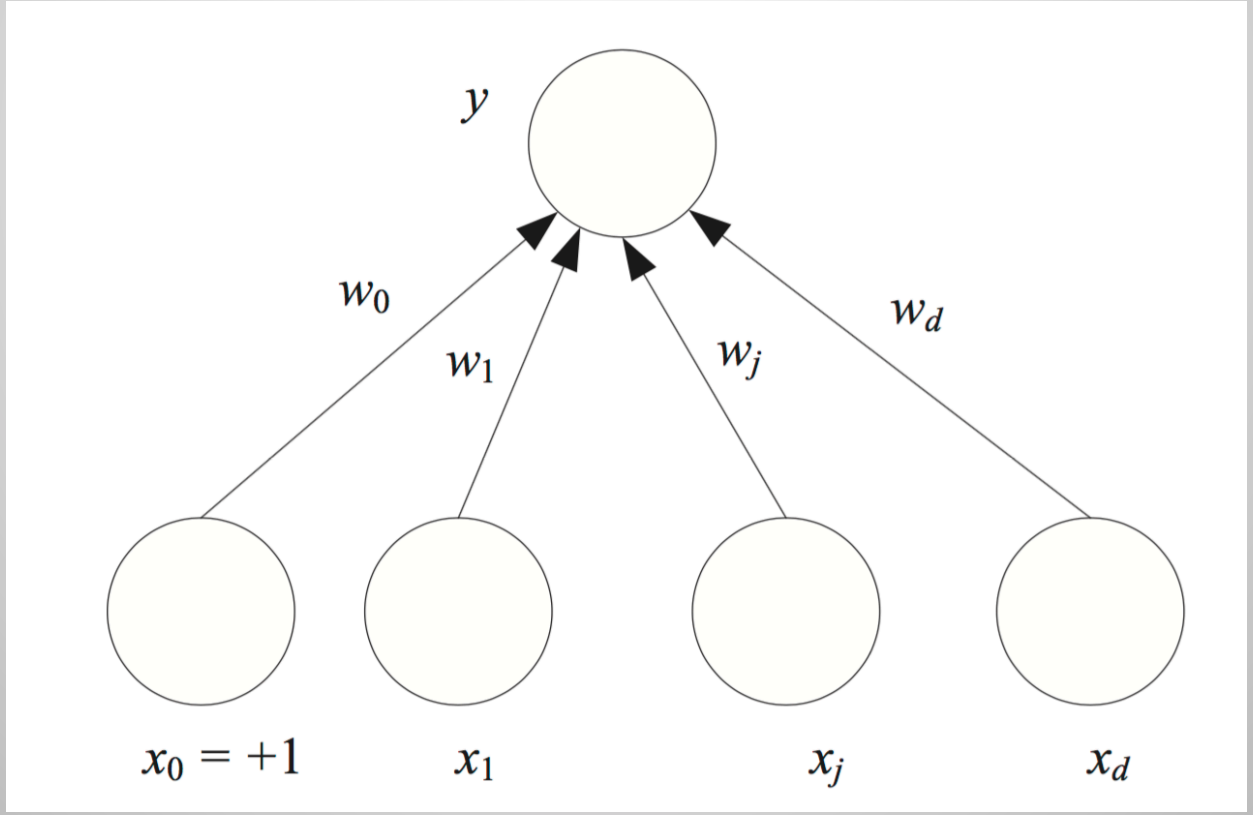
$$0 \underset{1}{\overset{0}{\gtrless}} w_0 + \sum_i w_i X_i$$

(Linear Decision Boundary)



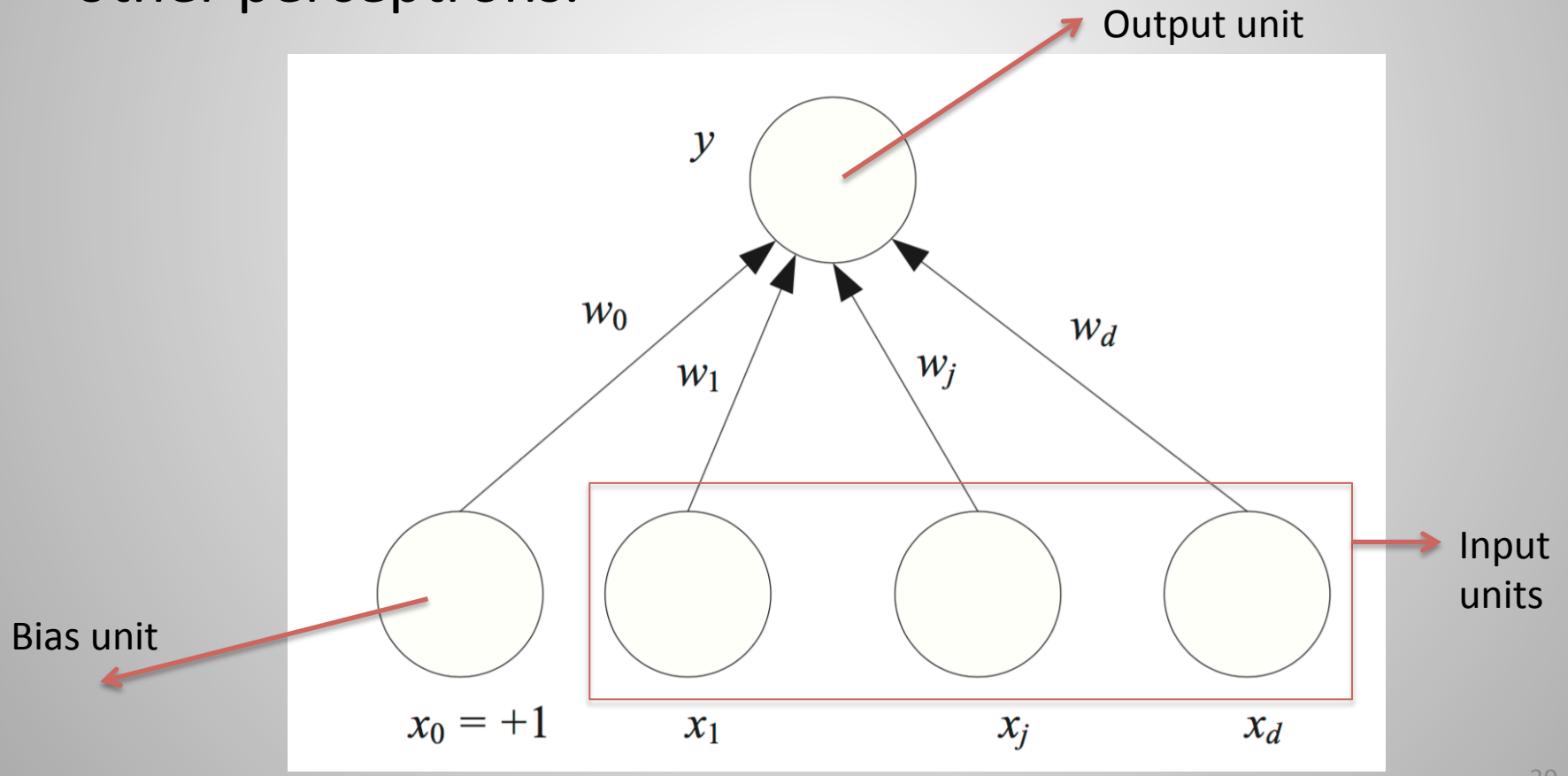
Perceptron

- is the basic processing element. It has inputs that may come from the environment or may be the outputs of other perceptrons.



Perceptron

- is the basic processing element. It has inputs that may come from the environment or may be the outputs of other perceptrons.

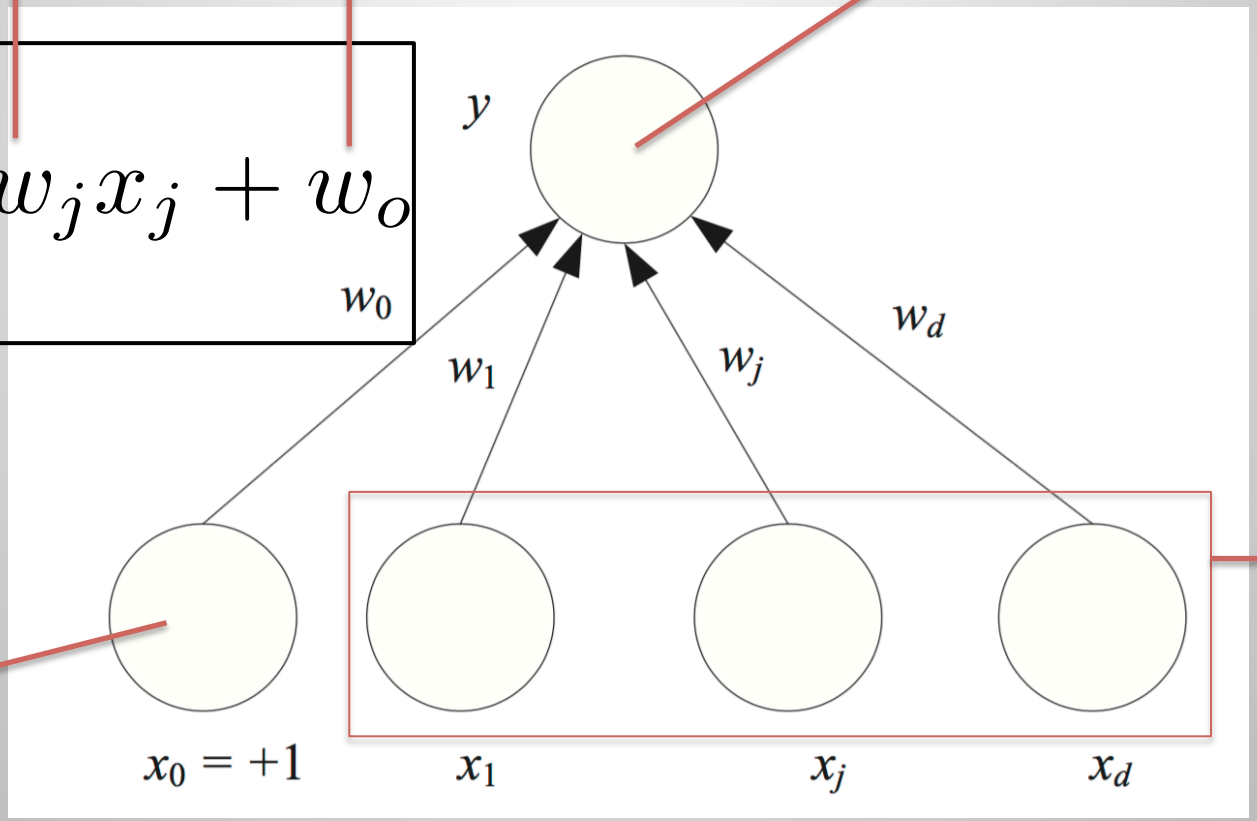


Perceptron

Connection weights

is the intercept value to make the model more general

$$y = \sum_{j=1}^{d} w_j x_j + w_o$$



Perceptron

$$y = \mathbf{w}^T \mathbf{x}$$

$$\mathbf{w} = [w_0, w_1, ..., w_d]^T$$

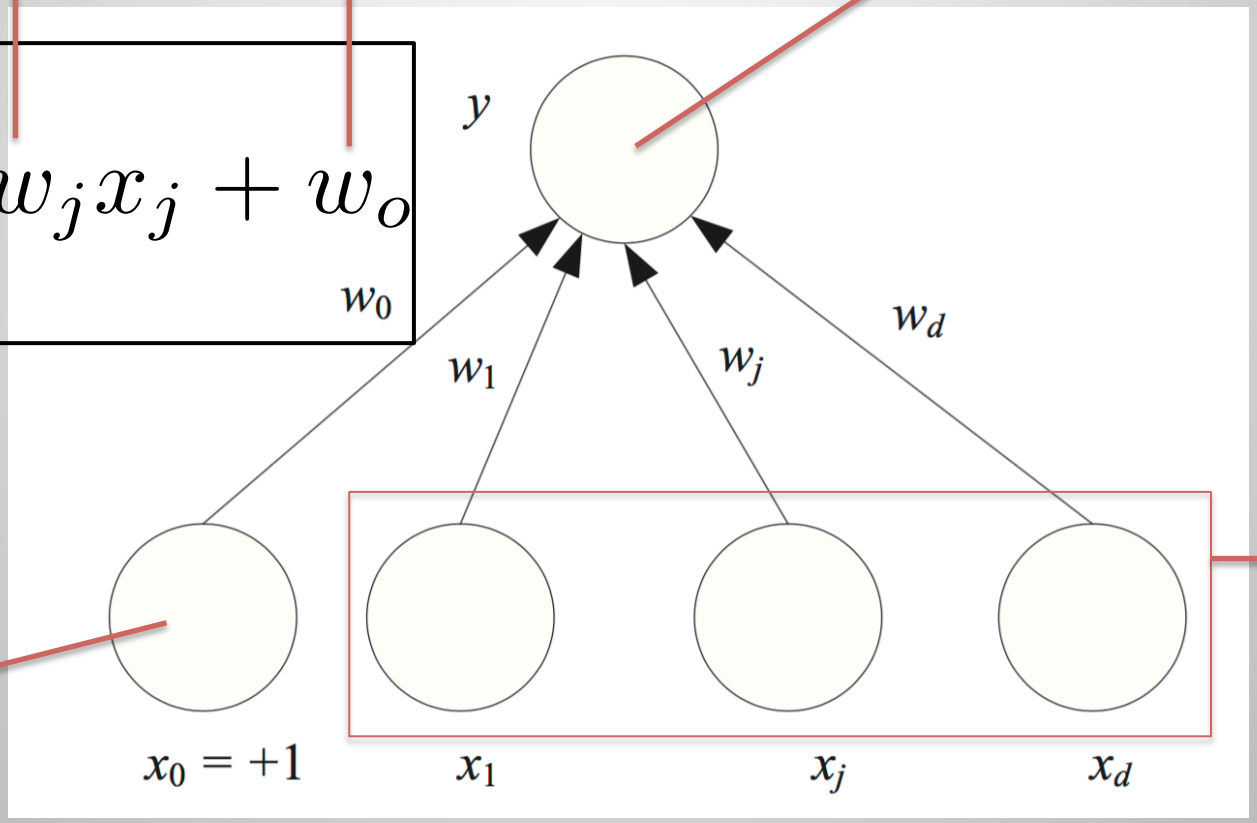
$$\mathbf{x} = [x_0, x_1, ..., x_d]^T$$

Connection weights

is the intercept value to make the model more general

$$y = \sum_{j=1}^{d} w_j x_j + w_o$$

w_0

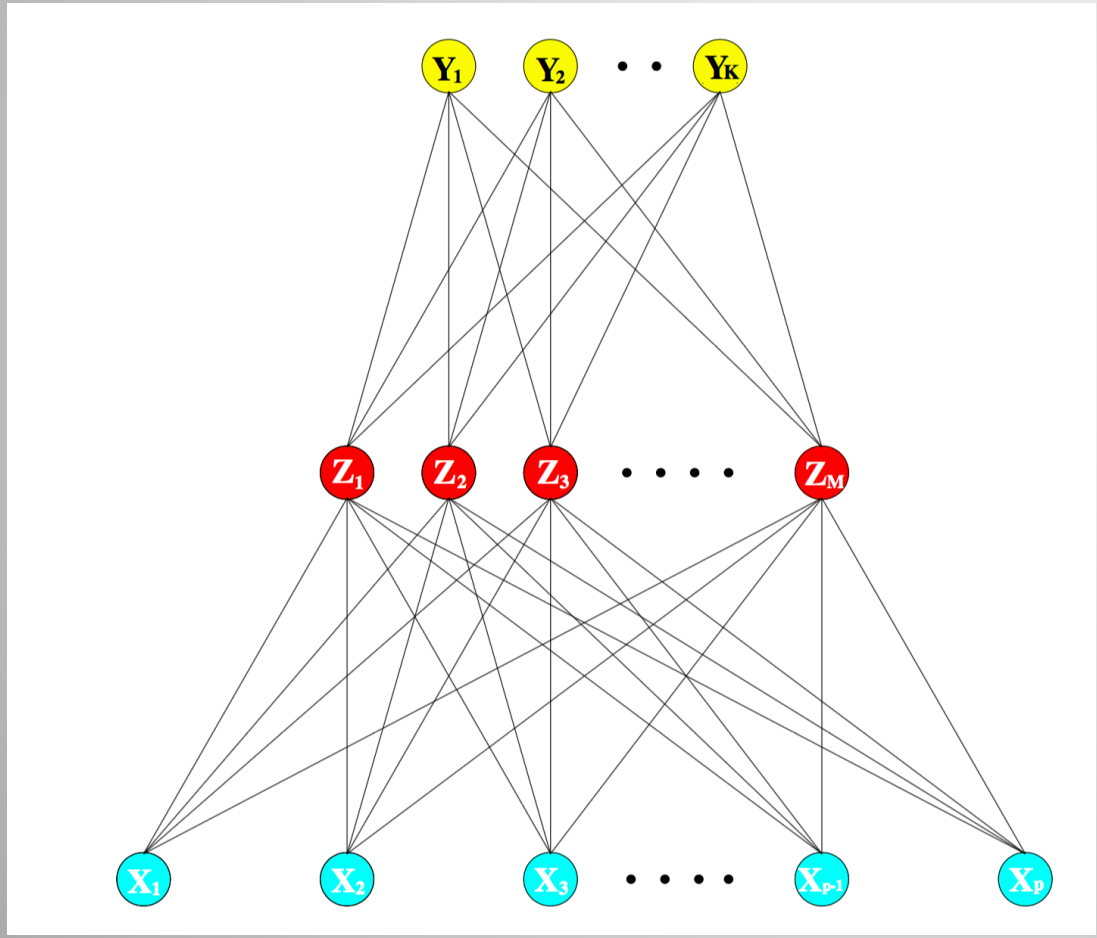


Perceptron-Nonlinear Statistical Models

- A NN is a two-stage regression or classification model

Perceptron-Nonlinear Statistical Models

- A NN is a two-stage regression or classification model

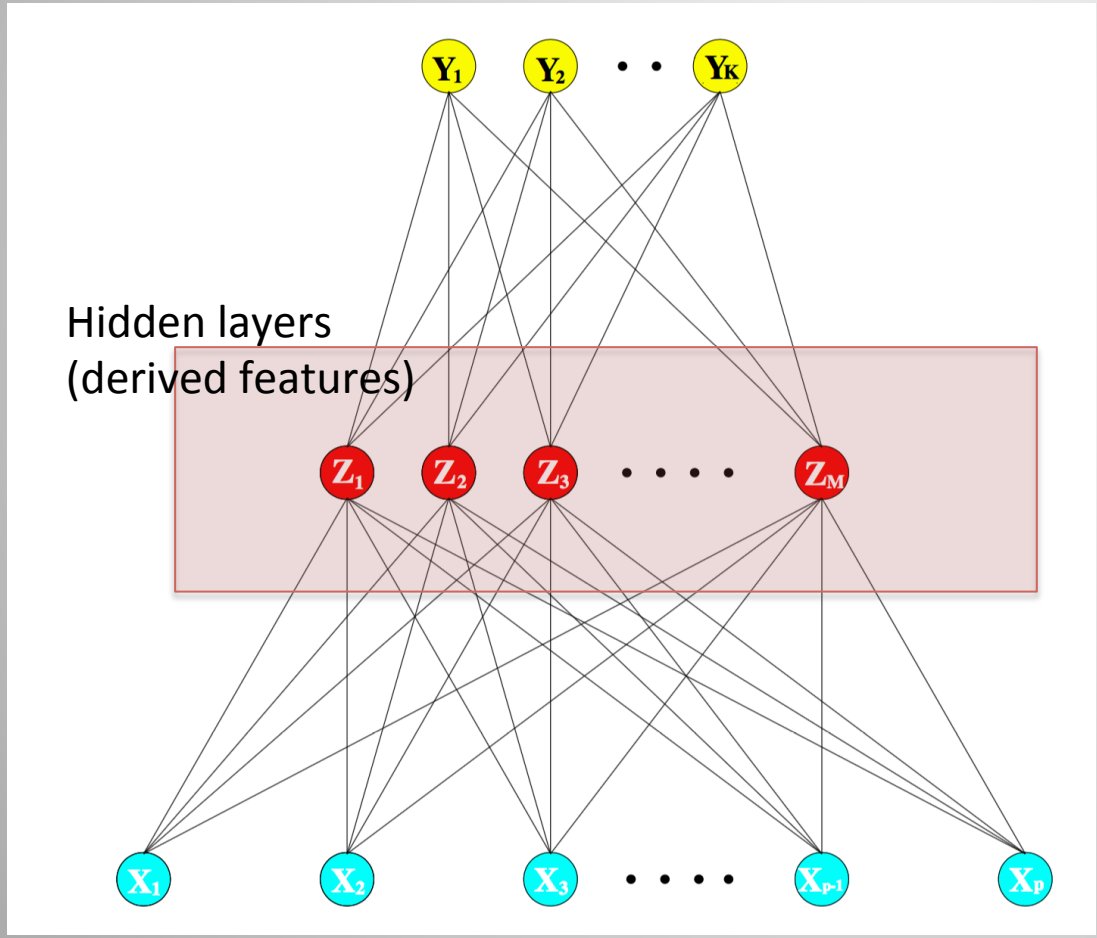


Typically K=1 for regression
And there is only one output
Unit at the top

For classification, K>1

Perceptron-Nonlinear Statistical Models

- A NN is a two-stage regression or classification model

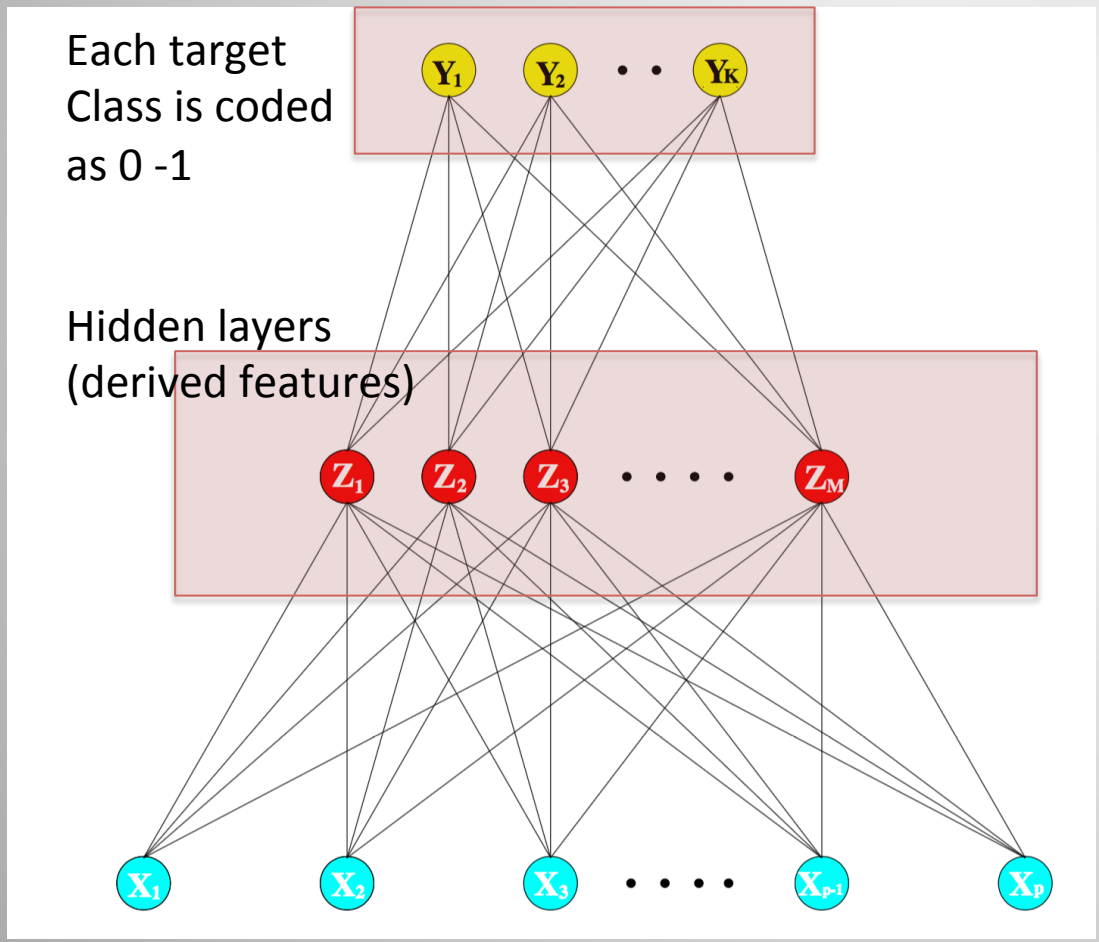


Typically K=1 for regression
And there is only one output
Unit at the top

For classification, K>1

Perceptron-Nonlinear Statistical Models

- A NN is a two-stage regression or classification model

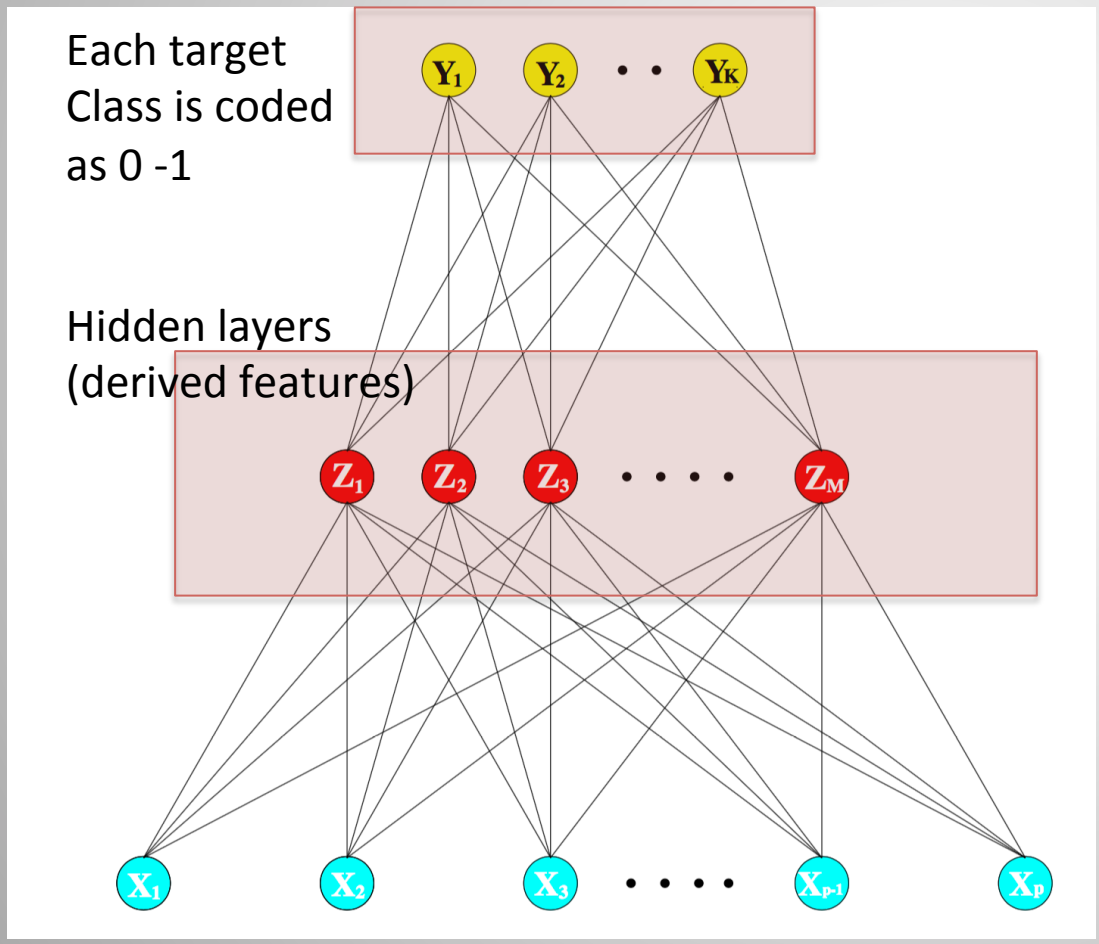


Typically K=1 for regression
And there is only one output
Unit at the top

For classification, K>1

Perceptron-Nonlinear Statistical Models

- A NN is a two-stage regression or classification model



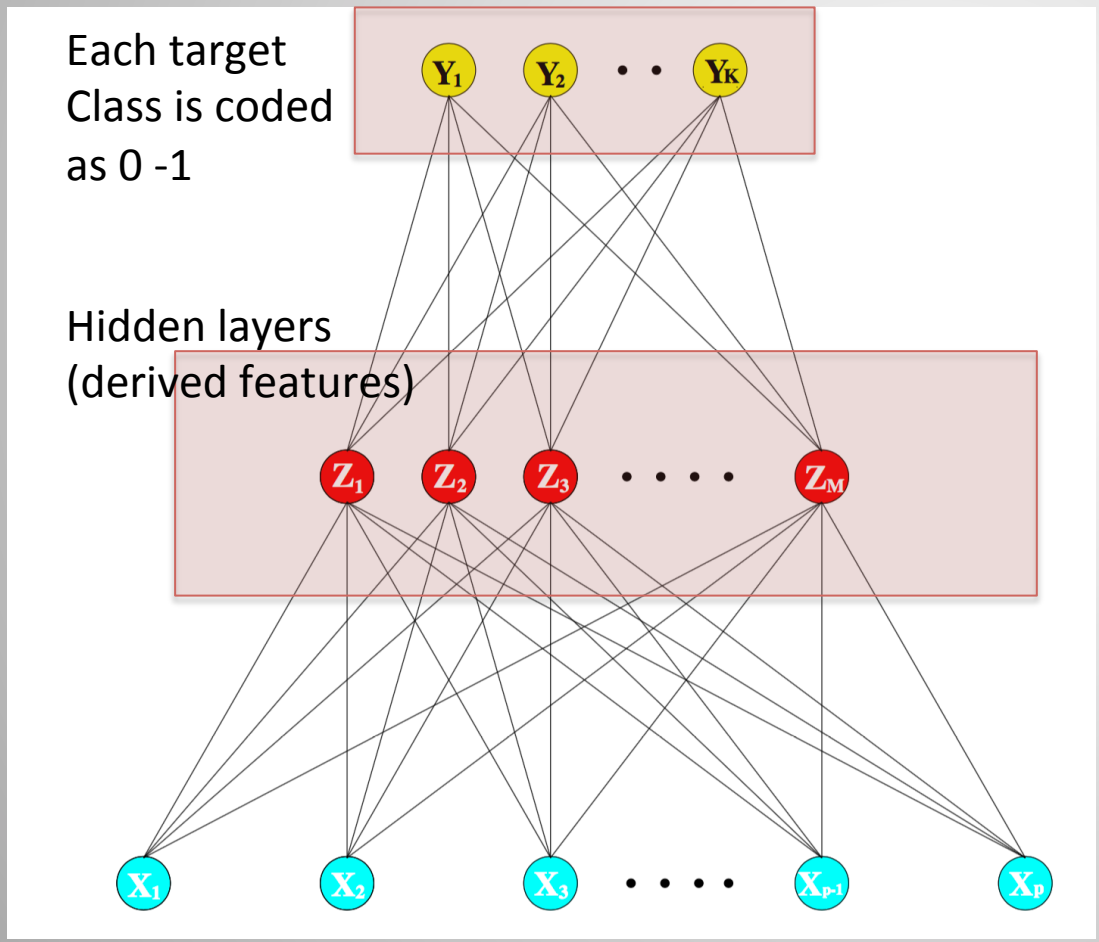
Typically K=1 for regression
And there is only one output
Unit at the top

For classification, K>1

$$Z_m = \sigma(\alpha_{0_m} + \alpha_m^T X),\ m = 1, ..., M$$

Perceptron-Nonlinear Statistical Models

- A NN is a two-stage regression or classification model



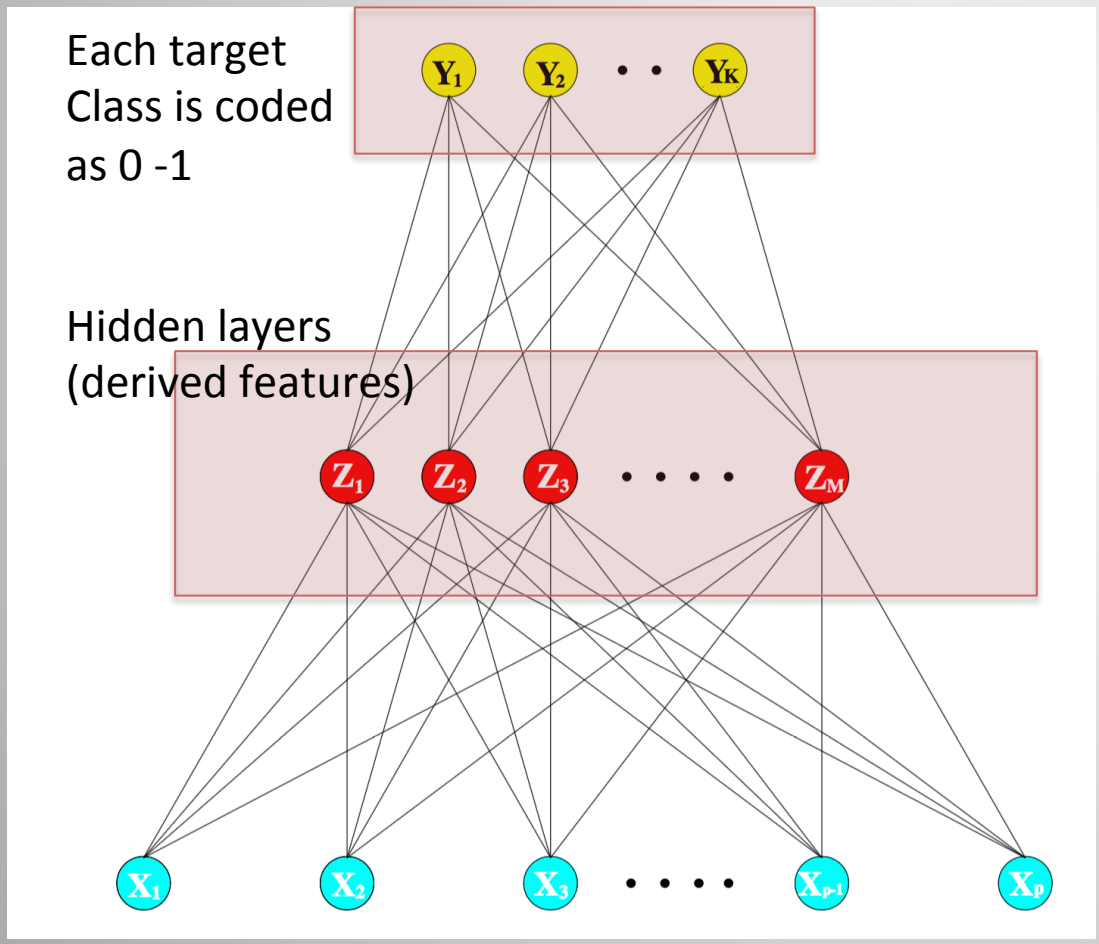
Typically K=1 for regression
And there is only one output
Unit at the top

For classification, K>1

$$Z_m = \sigma(\alpha_{0_m} + \alpha_m^T X),\ m = 1, ..., M$$
$$T_k = \beta_{0_k} + \beta_k^T Z,\ k = 1, ..., K$$

Perceptron-Nonlinear Statistical Models

- A NN is a two-stage regression or classification model



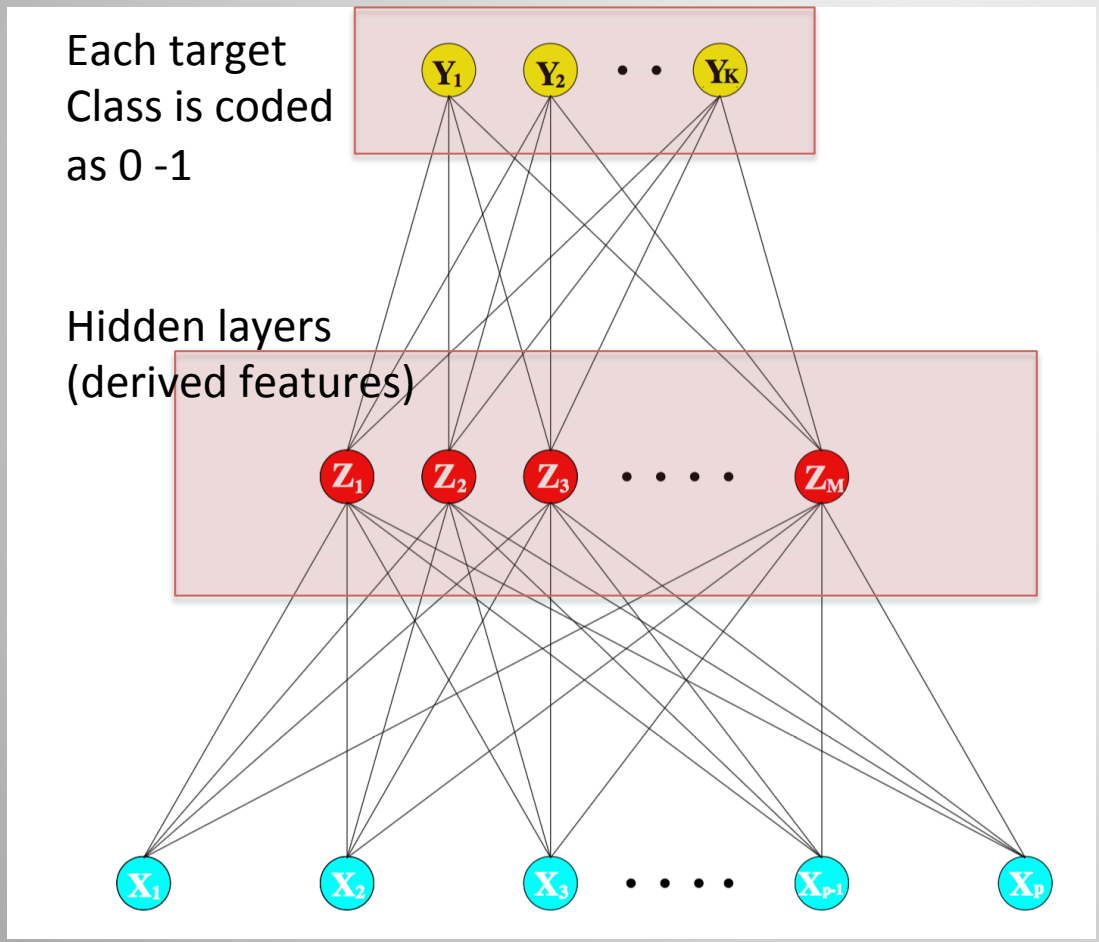
Typically K=1 for regression
And there is only one output
Unit at the top

For classification, K>1

$$Z_m = \sigma(\alpha_{0_m} + \alpha_m^T X),\ m = 1, ..., M$$
$$T_k = \beta_{0_k} + \beta_k^T Z,\ k = 1, ..., K$$
$$f_k(X) = g_k(T),\ k = 1, ..., K$$

Perceptron-Nonlinear Statistical Models

- A NN is a two-stage regression or classification model



For regression, output function is

$$g_k(T) = T_k$$

For classification, it is softmax func:

$$g_k(T) = \frac{e^{T_k}}{\sum_{l=1}^{K} e^{T_l}}$$

$$Z_m = \sigma(\alpha_{0_m} + \alpha_m^T X),\ m = 1, ..., M$$

$$T_k = \beta_{0_k} + \beta_k^T Z,\ k = 1, ..., K$$

$$f_k(X) = g_k(T),\ k = 1, ..., K$$

$$\sigma(v) = 1/(1 + e^{-v})$$

Fitting NNs

- The NN model has unknown parameters, often called weights, and we seek values for them that make the model fit the training data well.

$$\{\alpha_{0m}, \alpha_m;\ m = 1, 2, \ldots, M\}\ \ M(p+1) \text{ weights},$$
$$\{\beta_{0k}, \beta_k;\ k = 1, 2, \ldots, K\}\ \ K(M+1) \text{ weights}.$$

Fitting NNs

- The NN model has unknown parameters, often called weights, and we seek values for them that make the model fit the training data well.

$$\{\alpha_{0m}, \alpha_m;\ m = 1, 2, \ldots, M\}\ \ M(p+1) \text{ weights},$$
$$\{\beta_{0k}, \beta_k;\ k = 1, 2, \ldots, K\}\ \ K(M+1) \text{ weights}.$$

- For regression, we use sum-of-squared errors as our measure of fit:

$$R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2.$$

Fitting NNs

- The NN model has unknown parameters, often called weights, and we seek values for them that make the model fit the training data well.

$$\{\alpha_{0m}, \alpha_m;\ m = 1, 2, \ldots, M\}\ \ M(p+1) \text{ weights},$$
$$\{\beta_{0k}, \beta_k;\ k = 1, 2, \ldots, K\}\ \ K(M+1) \text{ weights}.$$

- For regression, we use sum-of-squared errors as our measure of fit:

$$R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2.$$

- For classification, we use either squared error or cross-entropy:

$$R(\theta) = -\sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \log f_k(x_i),$$

and the corresponding classifier is

$$G(x) = \text{argmax}_k f_k(x).$$

Fitting NN: Back-propagation

- Typically we don't want the global minimizer of R(.), as this likely to be an overfit solution.
- Instead, some regularization is needed:
  - This is achieved directly through a penalty term, or indirectly by early stopping.

Fitting NN: Back-propagation

- Typically we don't want the global minimizer of R(.), as this likely to be an overfit solution.
- Instead, some regularization is needed:
  - This is achieved directly through a penalty term, or indirectly by early stopping.
- The generic approach for minimizing R(.) is by gradient-descent, called back-propagation.

Fitting NN: Back-propagation

- Typically we don't want the global minimizer of R(.), as this likely to be an overfit solution.
- Instead, some regularization is needed:
  - This is achieved directly through a penalty term, or indirectly by early stopping.
- The generic approach for minimizing R(.) is by gradient-descent, called back-propagation.
- Due to compositional form of the model, the gradient can be easily derived using the chain rule for differentiation. This can be computed by a forward and backward sweep over the network, keeping track only of quantities local to each unit.

Back-propagation

$$z_{m_i} = \sigma(\alpha_{0_m} + \alpha_m^T x_i), \quad z_i = (z_{1i}, z_{2i}, ..., z_{Mi})$$

$$\begin{aligned} R(\theta) &\equiv \sum_{i=1}^{N} R_i \\ &= \sum_{i=1}^{N} \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2, \end{aligned}$$

Back-propagation

$$z_{m_i} = \sigma(\alpha_{0_m} + \alpha_m^T x_i), \quad z_i = (z_{1i}, z_{2i}, ..., z_{Mi})$$

$$\begin{aligned} R(\theta) &\equiv \sum_{i=1}^{N} R_i \\ &= \sum_{i=1}^{N} \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2, \end{aligned}$$

Taking the derivative of R w.r.t weight parameters

$$\frac{\partial R_i}{\partial \beta_{km}} = -2(y_{ik} - f_k(x_i)) g_k'(\beta_k^T z_i) z_{mi},$$

$$\frac{\partial R_i}{\partial \alpha_{m\ell}} = -\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i)) g_k'(\beta_k^T z_i) \beta_{km} \sigma'(\alpha_m^T x_i) x_{i\ell}.$$

Back-propagation

- Given these derivatives, a gradient descent update at the (r+1)st iteration has the form

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}},$$

$$\alpha_{m\ell}^{(r+1)} = \alpha_{m\ell}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{m\ell}^{(r)}},$$

Back-propagation

- Given these derivatives, a gradient descent update at the (r+1)st iteration has the form

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}},$$

$$\alpha_{m\ell}^{(r+1)} = \alpha_{m\ell}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{m\ell}^{(r)}},$$

Learning rate

Back-propagation

- Given these derivatives, a gradient descent update at the (r+1)st iteration has the form

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}},$$

$$\alpha_{m\ell}^{(r+1)} = \alpha_{m\ell}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{m\ell}^{(r)}},$$

Learning rate

$$\frac{\partial R_i}{\partial \beta_{km}} = \delta_{ki} z_{mi},$$

$$\frac{\partial R_i}{\partial \alpha_{m\ell}} = s_{mi} x_{i\ell}.$$

Errors from the current model at the output units

Errors from the current model at the hidden units

Back-propagation

- Given these derivatives, a gradient descent update at the (r+1)st iteration has the form

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}},$$

$$\alpha_{m\ell}^{(r+1)} = \alpha_{m\ell}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{m\ell}^{(r)}},$$

Batch learning

$$\frac{\partial R_i}{\partial \beta_{km}} = \delta_{ki} z_{mi},$$

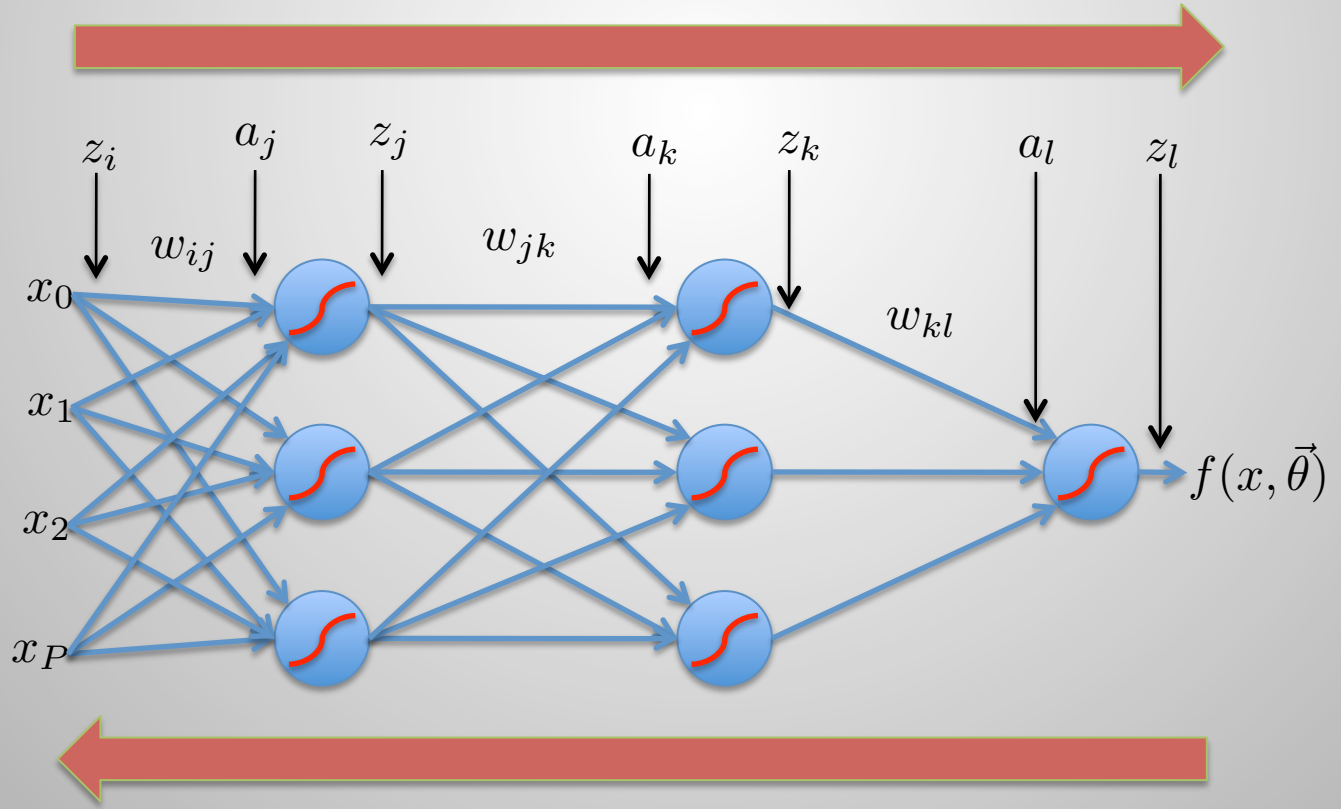
$$\frac{\partial R_i}{\partial \alpha_{m\ell}} = s_{mi} x_{i\ell}.$$

Backpropagation equations

$$s_{mi} = \sigma'(\alpha_m^T x_i) \sum_{k=1}^{K} \beta_{km} \delta_{ki},$$

Back-propagation-Delta Rule

1. Forward pass: calculate f(.) and obtain error rates,
2. Backward pass: calculate s from error rates, and update gradient equations.



Back-propagation-Delta Rule

1. Forward pass: calculate f(.) and obtain error rates,
2. Backward pass: calculate s from error rates, and update gradient equations.

If the weights are near zero: operative part of sigmoid is roughly linear, hence NN collapses into an approximately linear model.

Often, starting values are chosen to be random values near zero; hence the model starts nearly linear and becomes nonlinear as the weights increase.

Challenges in back-propagation?

- It requires labeled training data.
- The learning time does not scale well
  - It is slow in networks with multiple hidden layers.
- It can get stuck in poor local optima.

Avoid Overfitting

- NNs usually have too many weights, and will overfit the data at the global minimum of R.

Avoid Overfitting

- NNs usually have too many weights, and will overfit the data at the global minimum of R.
- **Weight decay:** is analogous to ridge regression used for linear models for avoiding overfit.

Avoid Overfitting

- NNs usually have too many weights, and will overfit the data at the global minimum of R.
- **Weight decay:** is analogous to ridge regression used for linear models for avoiding overfit.
- A new functional is to be minimized:

$$R(\theta) + \lambda J(\theta)$$

Avoid Overfitting

- NNs usually have too many weights, and will overfit the data at the global minimum of R.
- **Weight decay:** is analogous to ridge regression used for linear models for avoiding overfit.
- A new functional is to be minimized:

$$R(\theta) + \lambda J(\theta)$$

where

$$J(\theta) = \sum_{km} \beta_{km}^2 + \sum_{ml} \alpha_{ml}^2$$

Avoid Overfitting

- NNs usually have too many weights, and will overfit the data at the global minimum of R.
- **Weight decay:** is analogous to ridge regression used for linear models for avoiding overfit.
- A new functional is to be minimized:

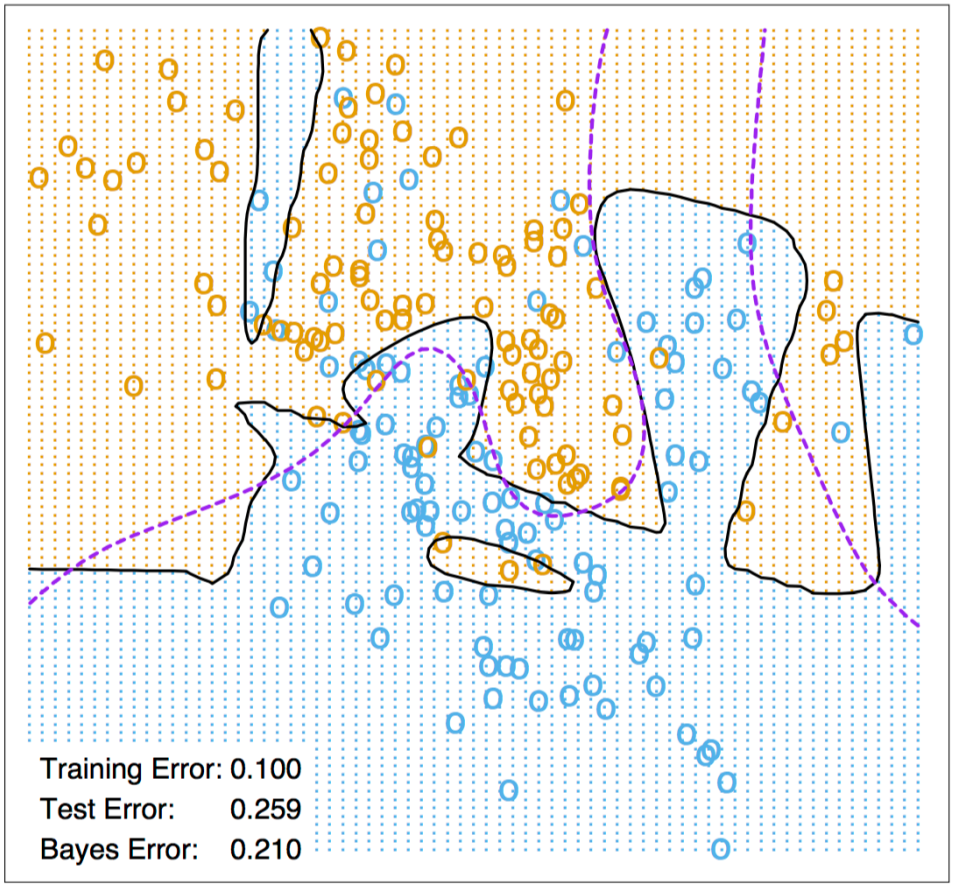
$$R(\theta) + \lambda J(\theta)$$

where

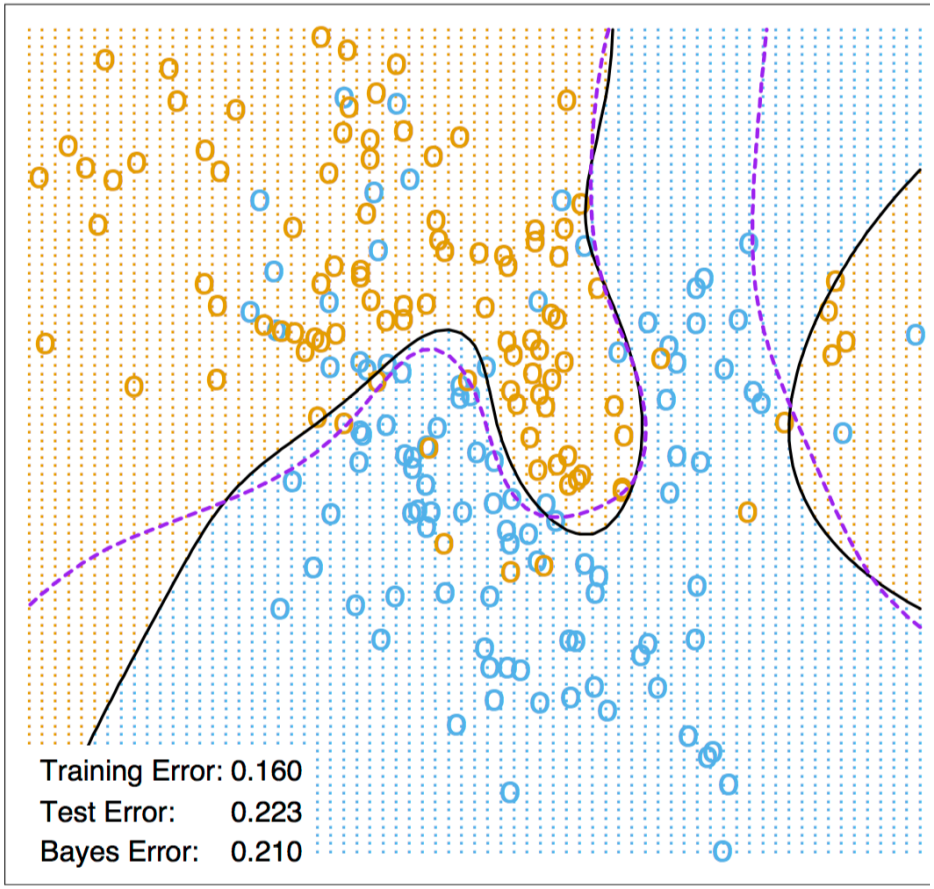
$$J(\theta) = \sum_{km} \beta_{km}^2 + \sum_{ml} \alpha_{ml}^2$$

where λ is tuning parameter >0, estimated by cross validation

Neural Network - 10 Units, No Weight Decay



Neural Network - 10 Units, Weight Decay=0.02

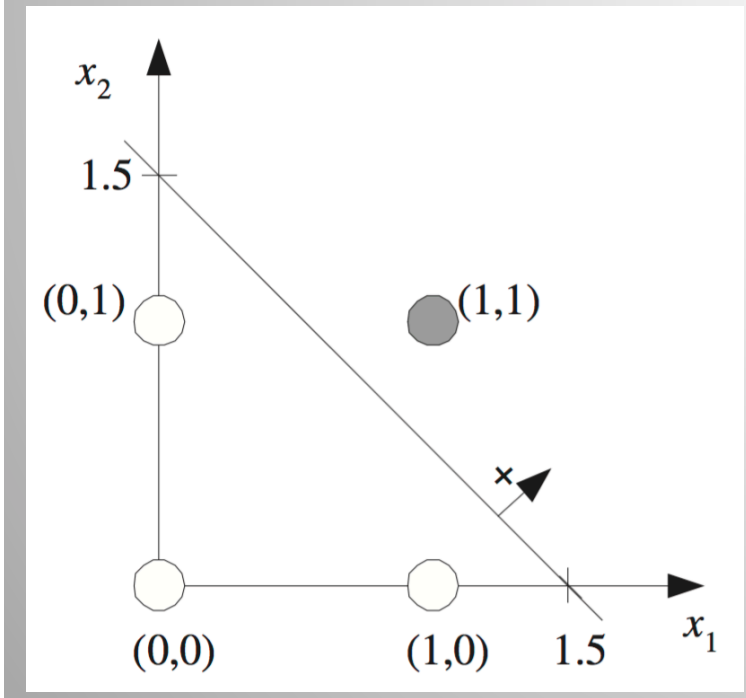


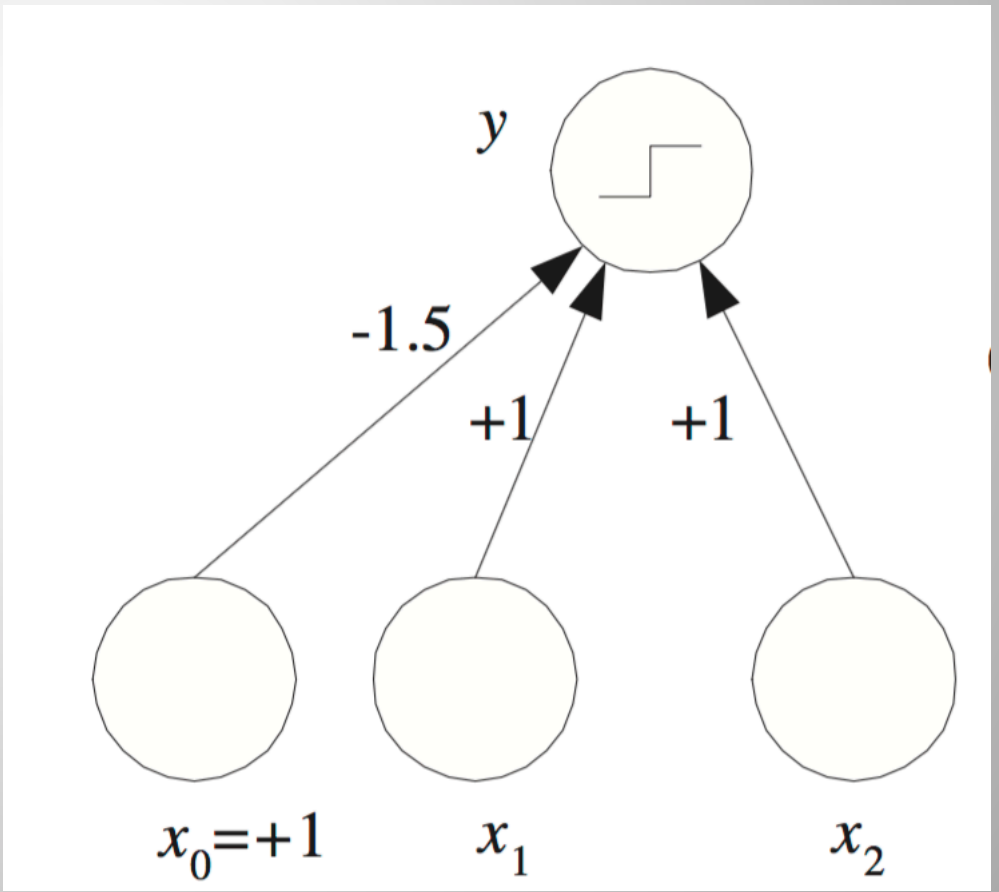
Ex: Learning Boolean Functions

AND

x_1	x_2	r
0	0	0
0	1	0
1	0	0
1	1	1

$$y = s(x_1 + x_2 - 1.5)$$





Question?

- What is the Boolean function that the following perceptron calculates?

$$y = s(-x + 0.5)$$

Question?

- What is the Boolean function that the following perceptron calculates?

$$y = s(-x + 0.5)$$

NOT of x

Question?

- What is the Boolean function that the following perceptron calculates?

$$y = s(x_1 + x_2 - 0.5)$$

Question?

- What is the Boolean function that the following perceptron calculates?

$$y = s(x_1 + x_2 - 0.5)$$

x1 OR x2

Key Ideas in Neural Nets

1. Learn features from data

Key Ideas in Neural Nets

1. Learn features from data
2. Use differentiable functions that produce features efficiently

Key Ideas in Neural Nets

1. Learn features from data
2. Use differentiable functions that produce features efficiently
3. End-to-End learning: no distinction between feature extractor and classifier

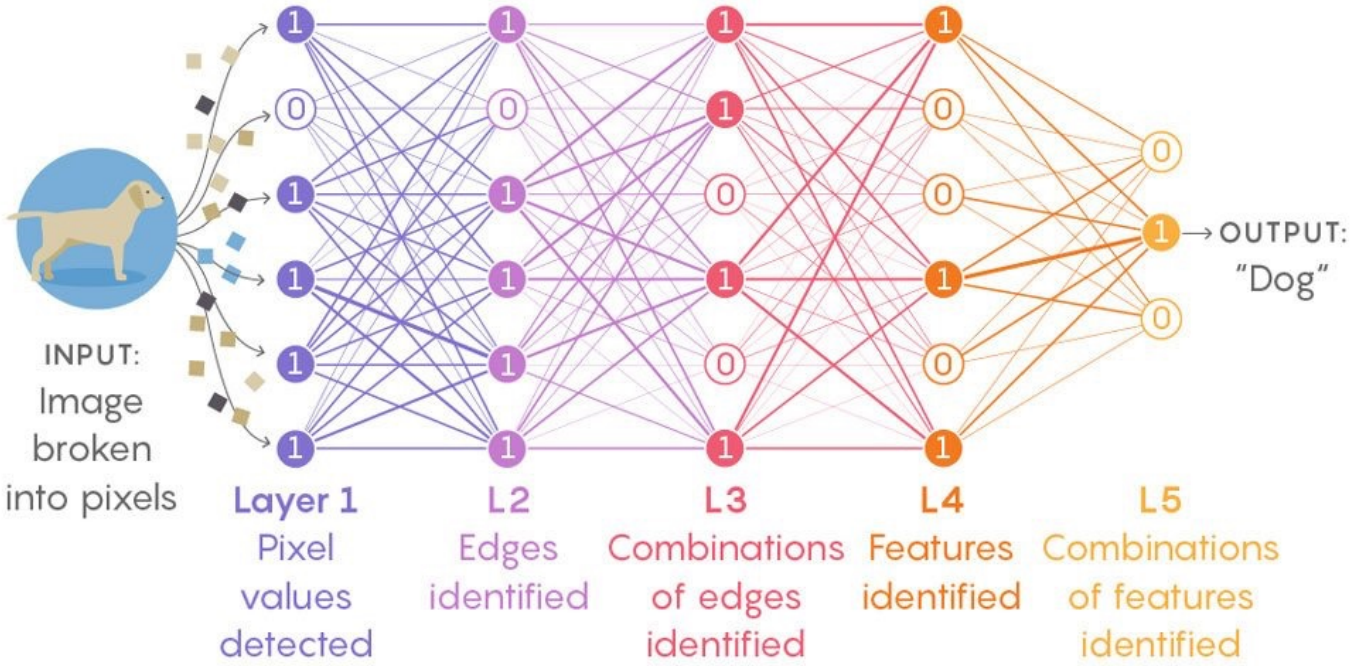
Key Ideas in Neural Nets

1. Learn features from data
2. Use differentiable functions that produce features efficiently
3. End-to-End learning: no distinction between feature extractor and classifier
4. “Deep” architectures: cascade of simpler non-linear modules

Toy Example: DL

Learning From Experience

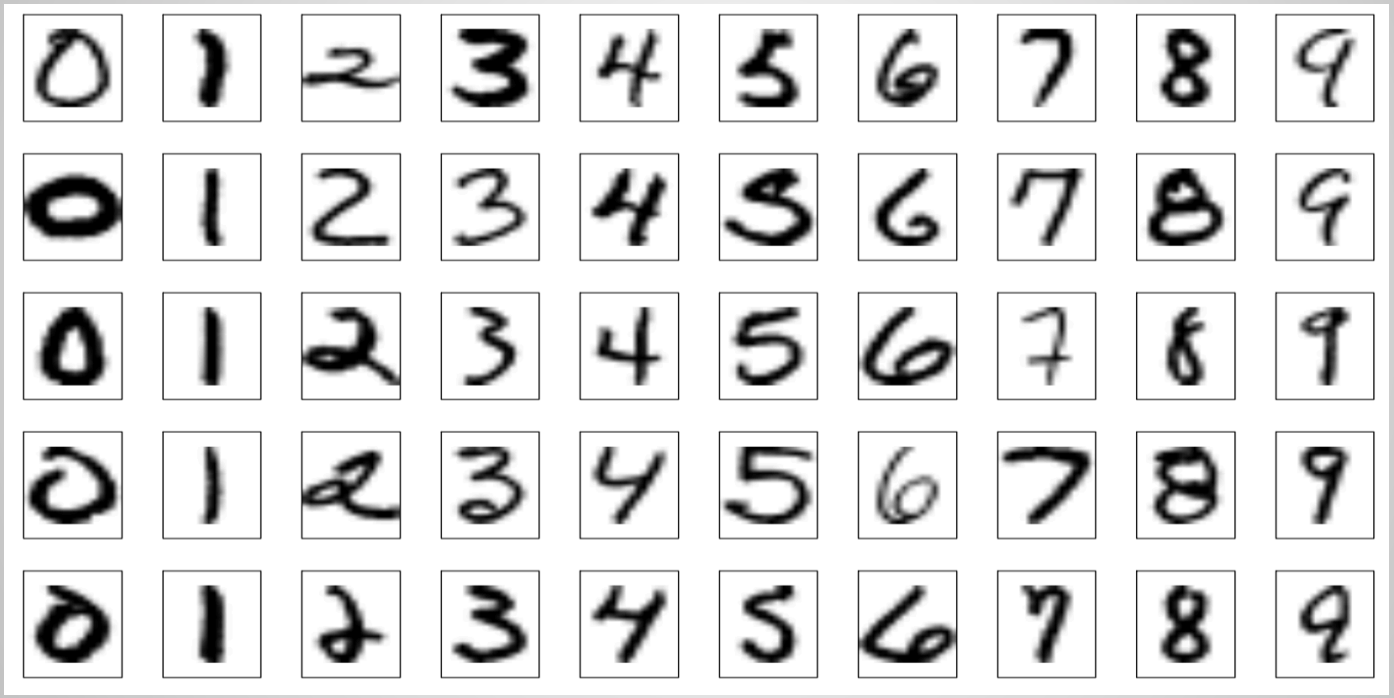
Deep neural networks learn by adjusting the strengths of their connections to better convey input signals through multiple layers to neurons associated with the right general concepts.



When data is fed into a network, each artificial neuron that fires (labeled "1") transmits signals to certain neurons in the next layer, which are likely to fire if multiple signals are received. The process filters out noise and retains only the most relevant features.

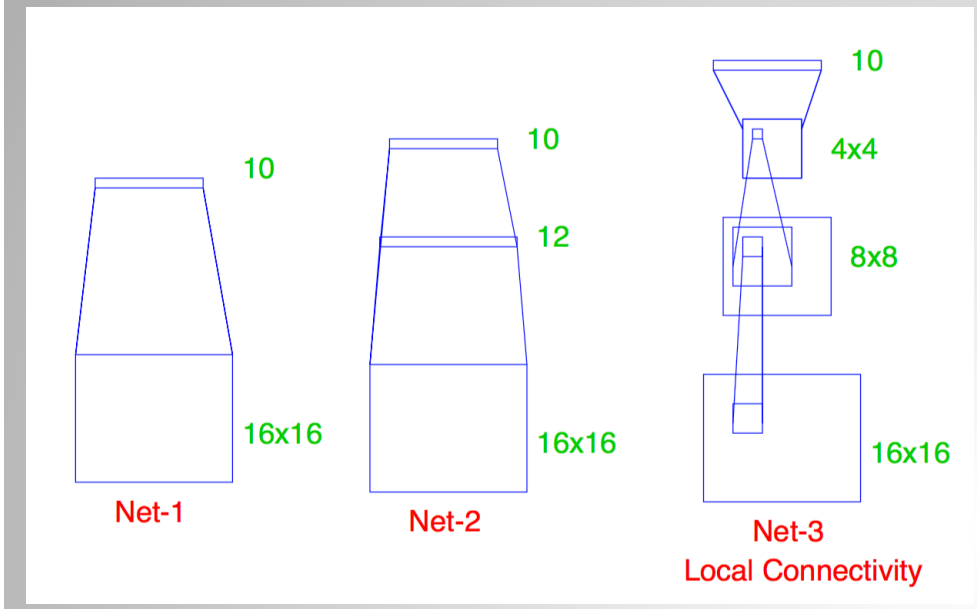
ZIP-CODE example

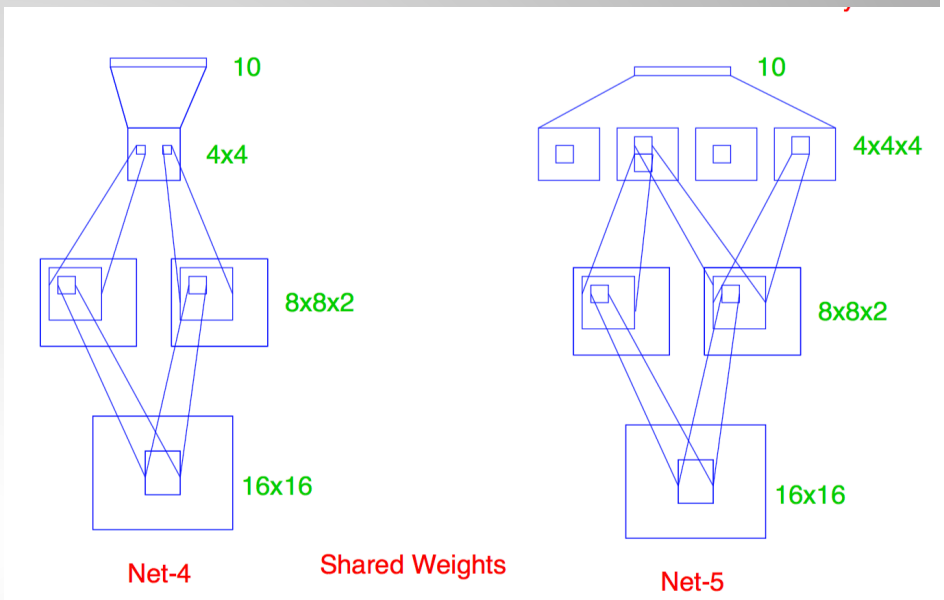
- Character recognition task: classification of handwritten numerals.
- It has remained a benchmark problem in the field for many years.



- Each image is 16x16 8-bit grayscale representation of handwritten digit.

5 networks used in the ZIP-CODE example





Net-1: No hidden layer, equivalent to multinomial logistic regression.

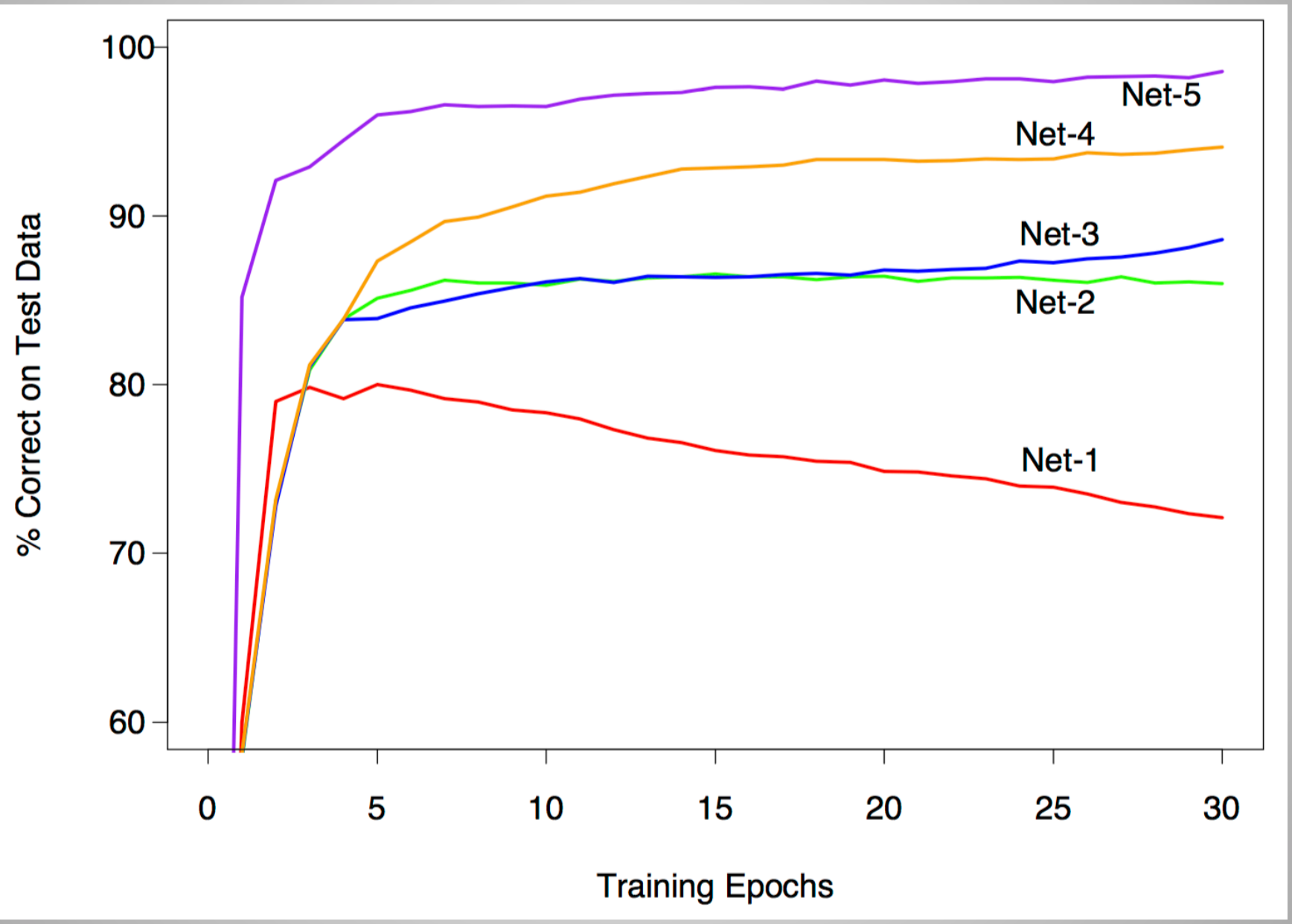
Net-2: One hidden layer, 12 hidden units fully connected.

Net-3: Two hidden layers locally connected.

Net-4: Two hidden layers, locally connected with weight sharing.

Net-5: Two hidden layers, locally connected, two levels of weight sharing.

5 networks used in the ZIP-CODE example



References and Slice Credits

- Rob Fergus, Deep Learning Tutorial CVPR 2012
- A. Torralba, MIT
- Aarti Singh & Barnabas Poczos (CMU)
- Hastie, Tibshirani, Friedman, The Elements of Statistical Learning.