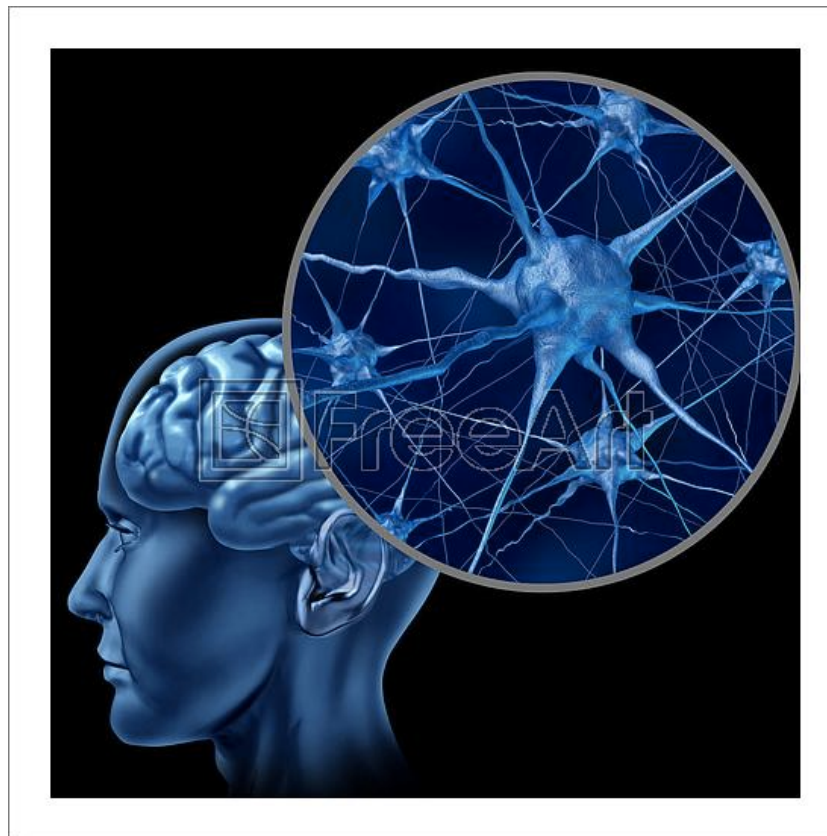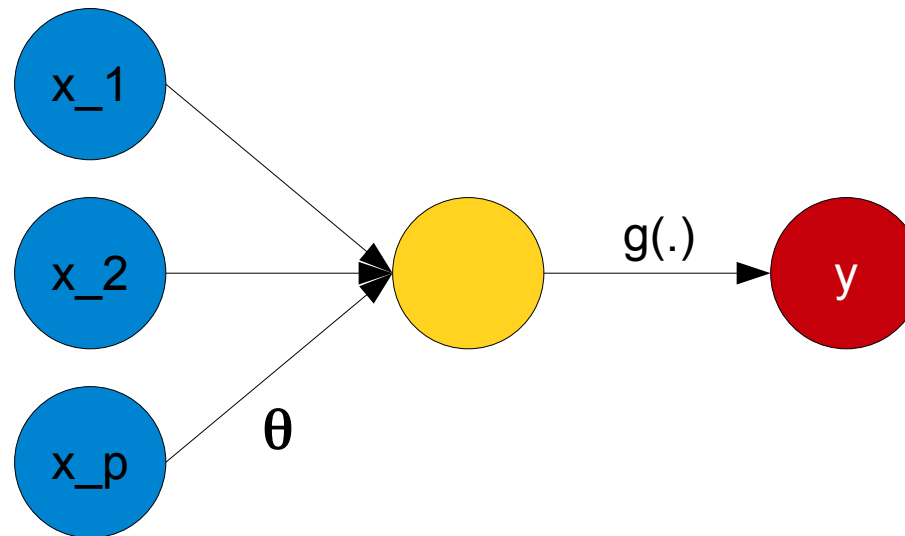# 機械学習特論
~理論とアルゴリズム~

(Neural Networks)

講師：西郷浩人

Neural Network; what is it ?

- Originally developed in 1950's by mimicking the network of neurons in human brain.
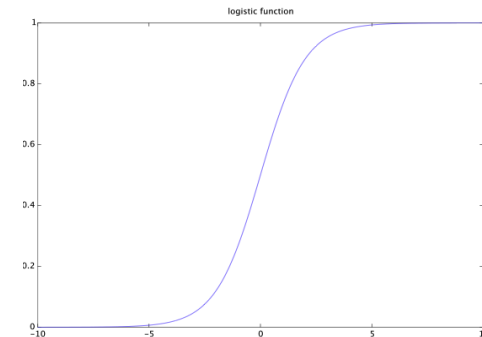
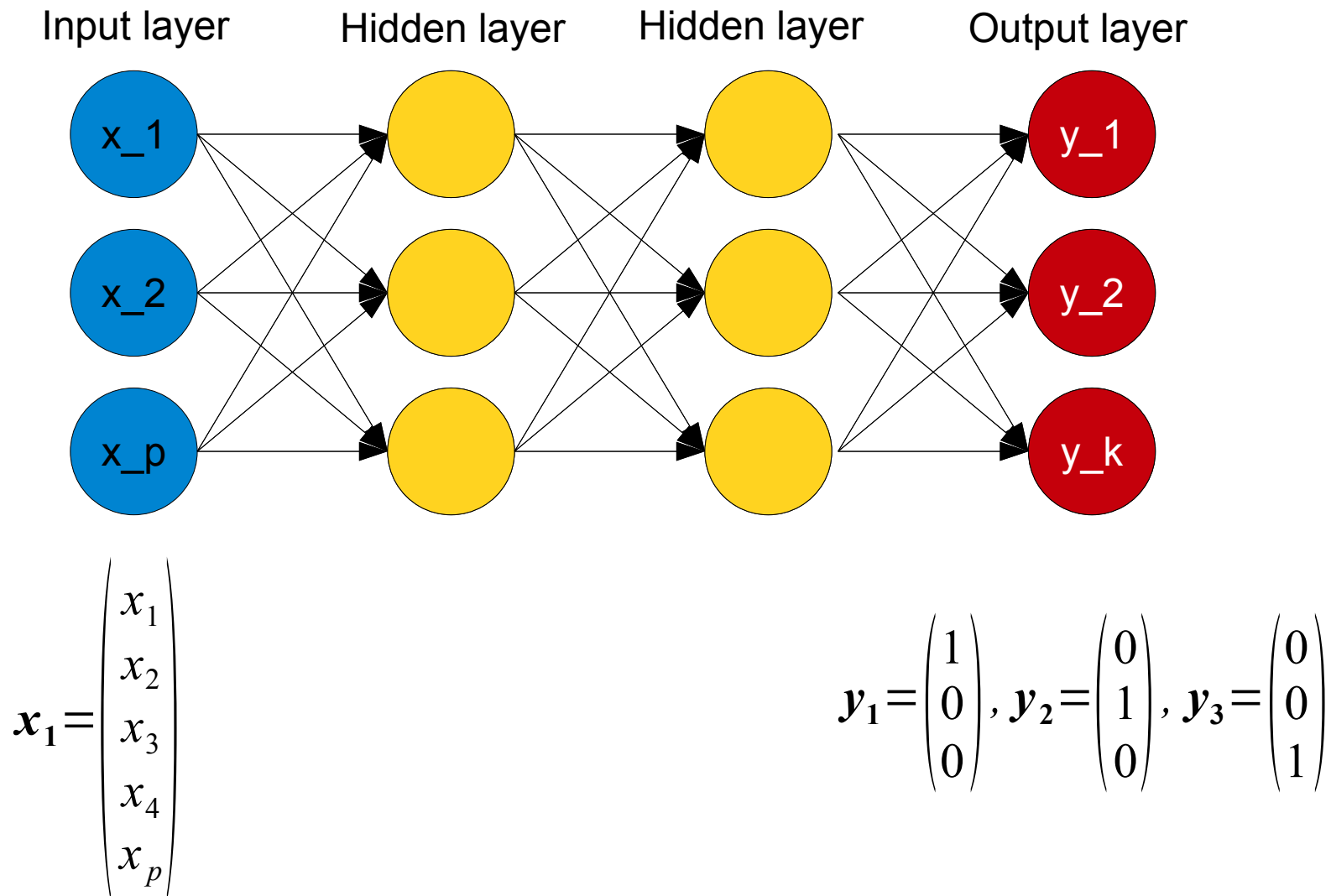# Single layer network
(a.k.a. linear / logistic regression)

$$y = g(x'\theta) = h_\Theta(x)$$

g(.) is a logistic sigmoid function

$$g(x) = \frac{\exp(-x)}{1 + \exp(-x)}$$

Multi-layer network
for multi-class classification

Input layer   Hidden layer   Hidden layer   Output layer

$$\boldsymbol{x_1} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_p \end{pmatrix}$$

$$\boldsymbol{y_1} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \boldsymbol{y_2} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \boldsymbol{y_3} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Multi-layer network model parameters

- a^(l): input to the l-th layer
- z^(l): output of the l-th layer

$$a^{(1)}=x$$

$$\Theta^{(1)}$$
$$z^{(2)}=\Theta^{(1)}a^{(1)}$$
$$a^{(2)}=g(z^{(2)})$$

$$\Theta^{(2)}$$
$$z^{(3)}=\Theta^{(2)}a^{(2)}$$
$$a^{(3)}=g(z^{(3)})$$

$$\Theta^{(3)}$$
$$z^{(4)}=\Theta^{(3)}a^{(3)}$$
$$a^{(4)}=g(z^{(4)})$$

$$h_{\Theta}(x)=a^{(4)}$$
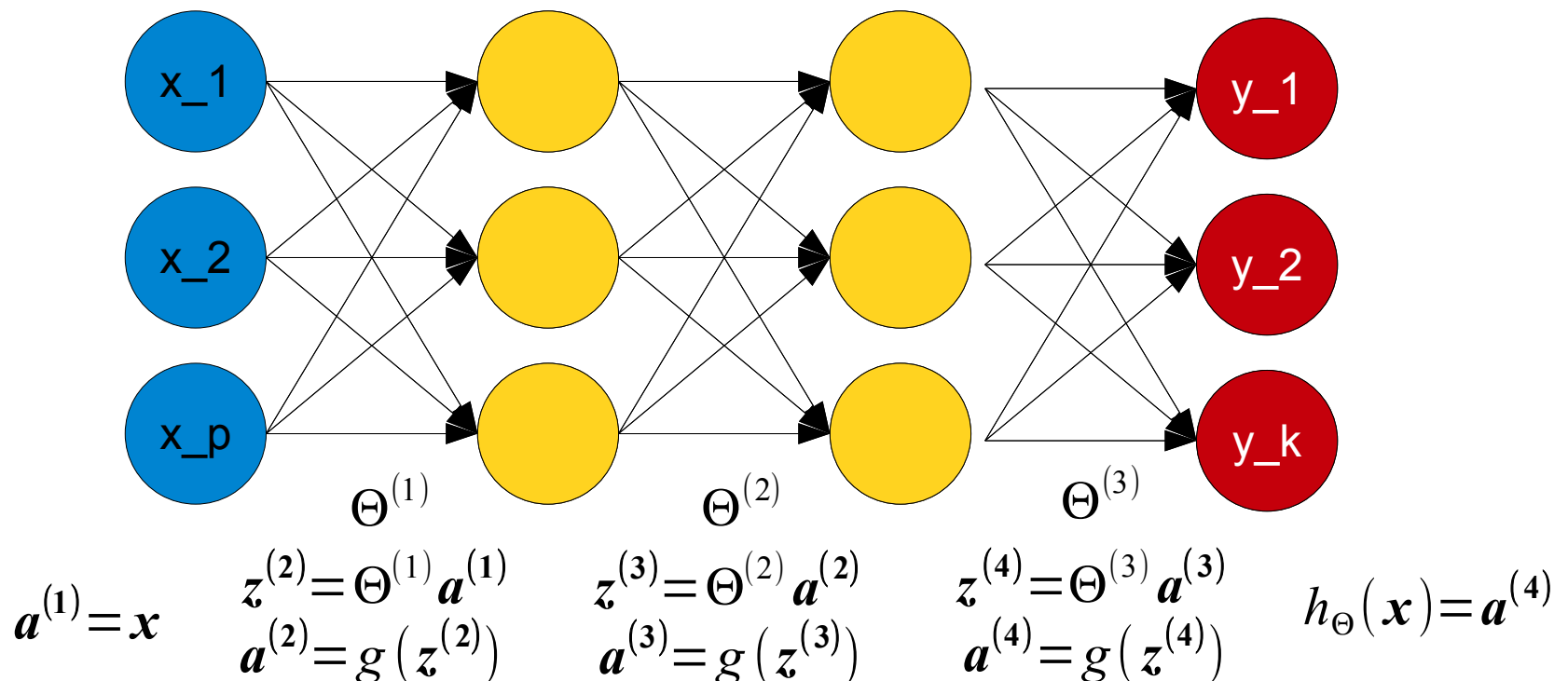
Difficulty in training multi-layer network

Input layer   Hidden layer   Hidden layer   Output layer

x_1 x_2 x_p $\Theta^{(1)}$ $\Theta^{(2)}$ $\Theta^{(3)}$ y_1 y_2 y_k
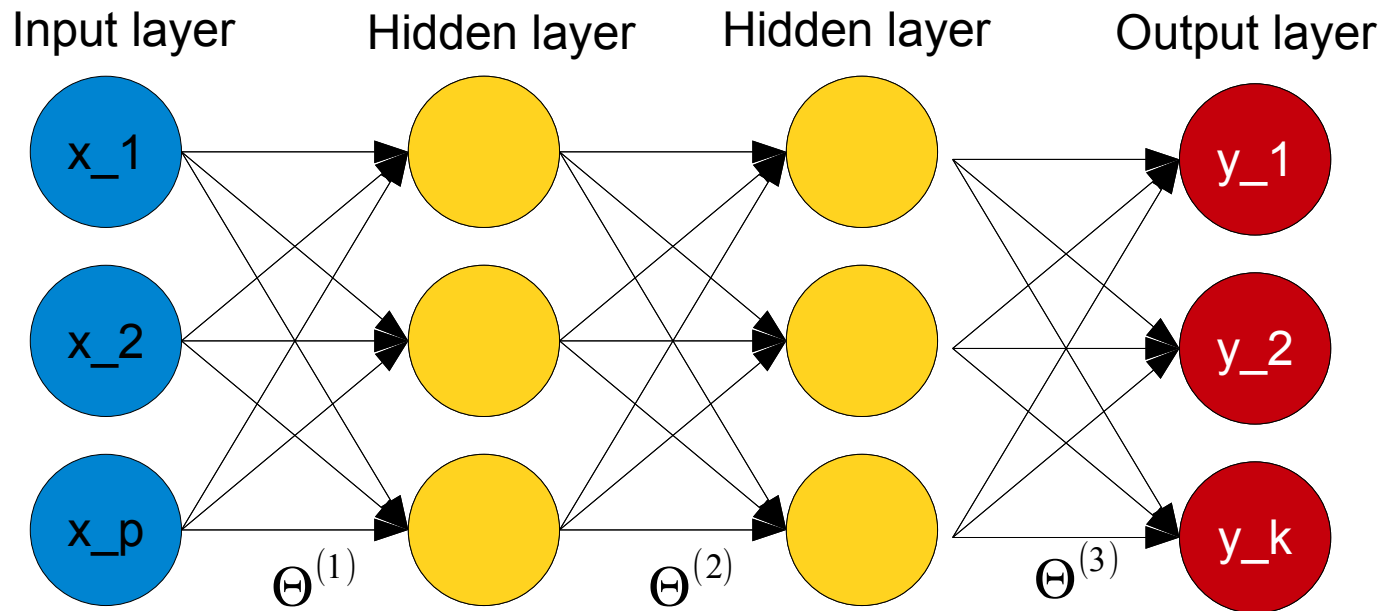
- Number of parameters in this example:

$$\left|\Theta^{(1)}\right|+\left|\Theta^{(2)}\right|+\left|\Theta^{(3)}\right|=3\times3+3\times3+3\times3=27$$

- More formerly: $p\times h_1+h_1\times h_2+h_2\times k$
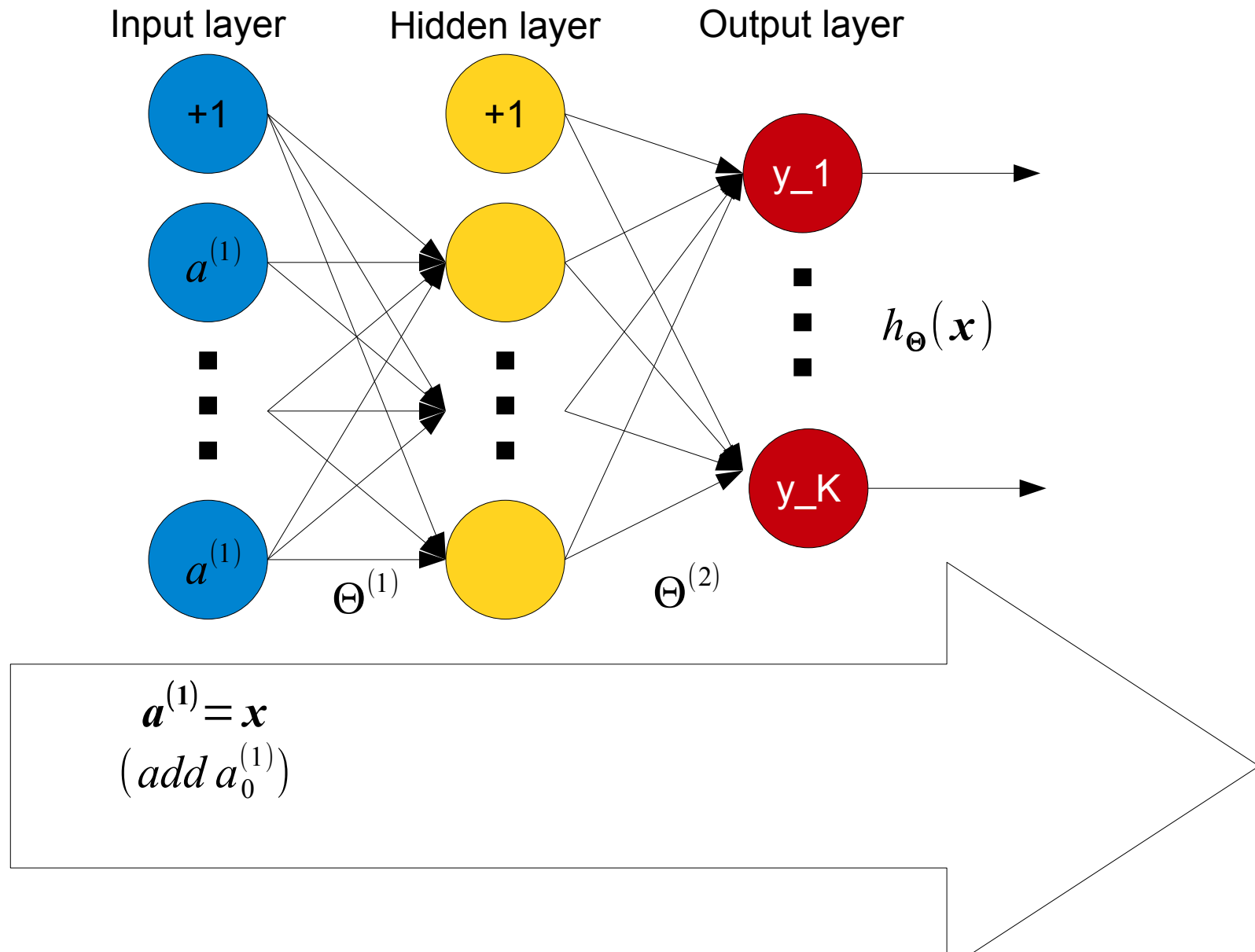
  - Easily above millions !

Efficiently updating parameters via forward/backward propagation

- For handling large number of parameters, an efficient updating technique is necessary.

- Forward propagation algorithm computes variables **a** and **z** from left (input) to right (output).

- Backward propagation algorithm computes δ from right (output) to left (input).
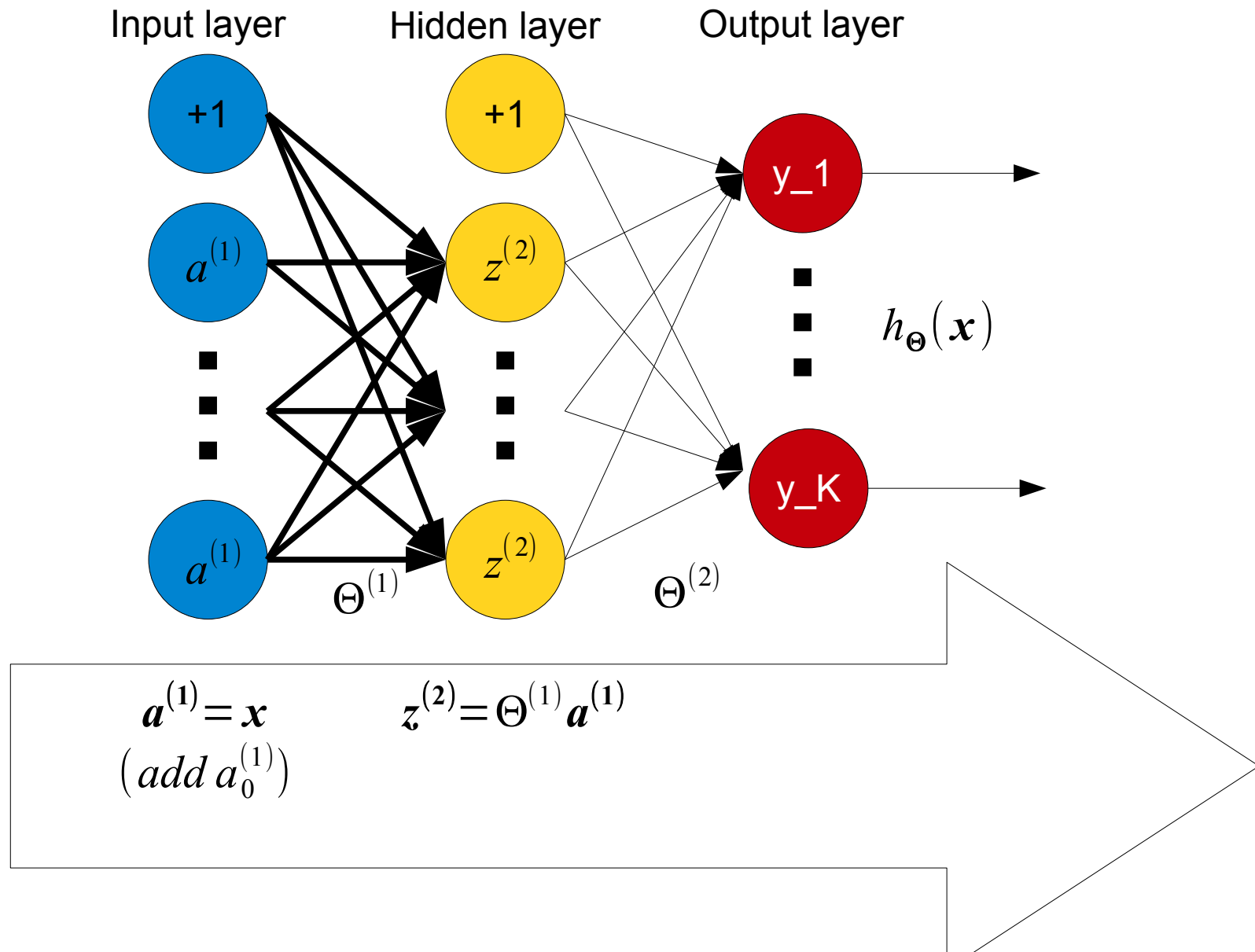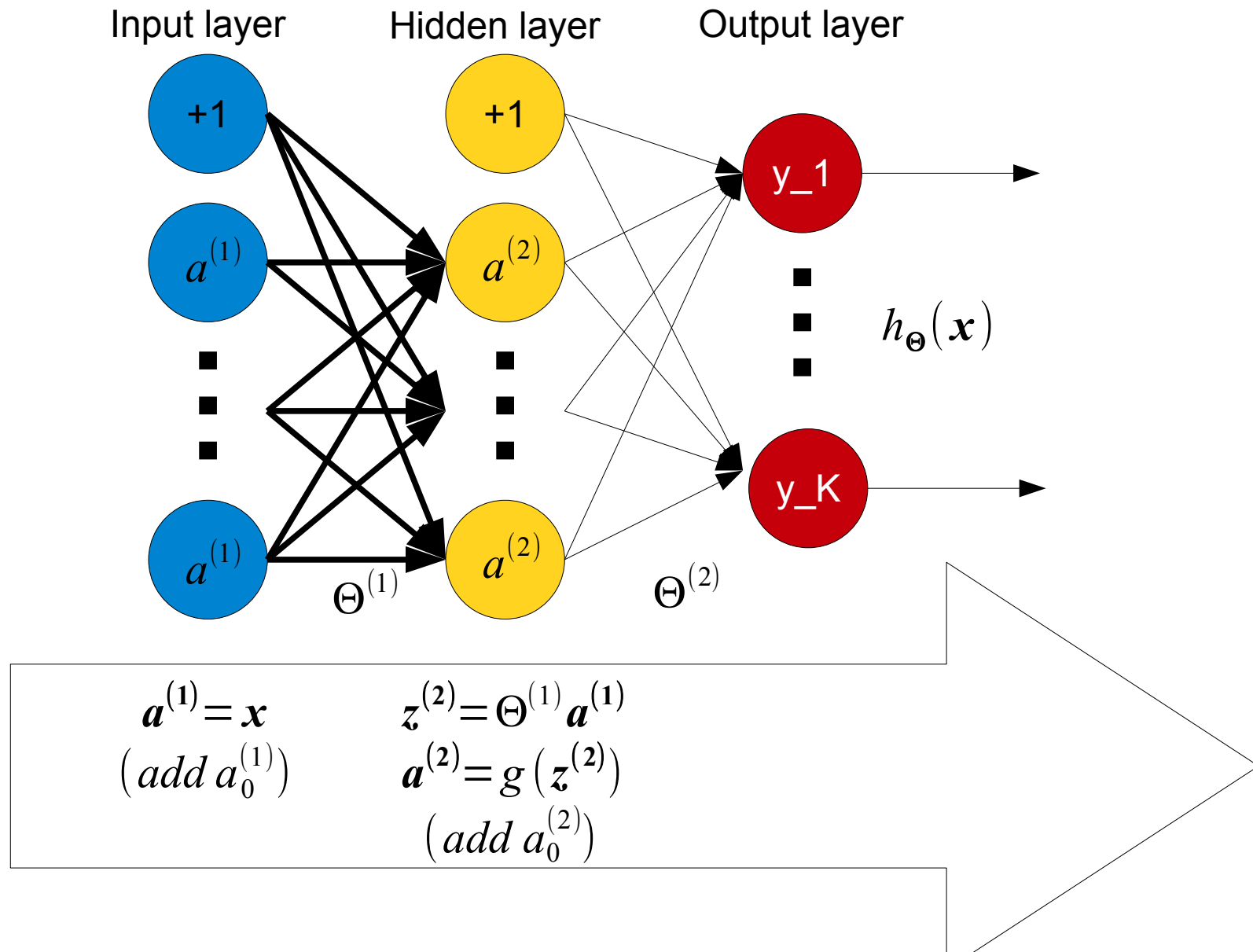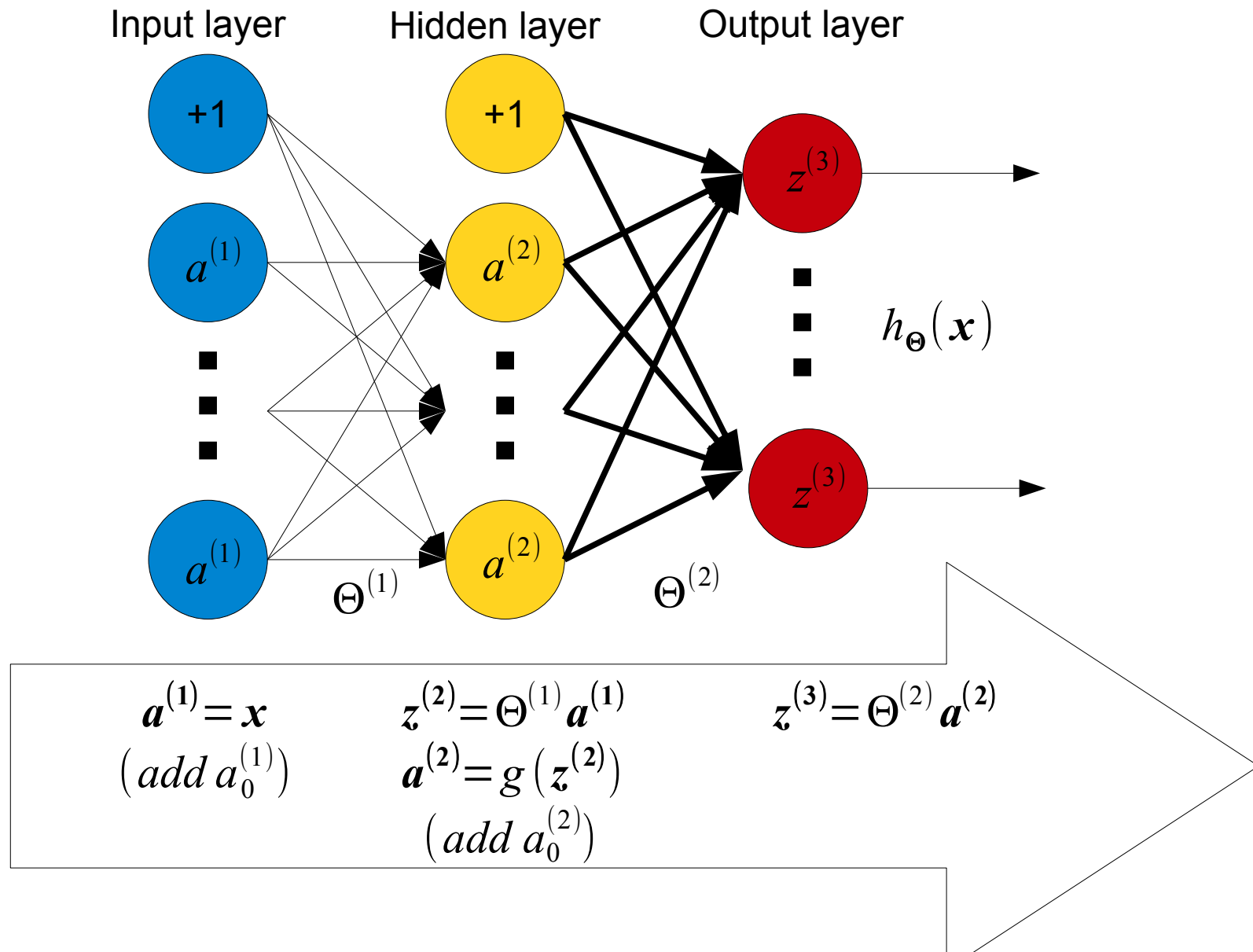
Feedforward Propagation
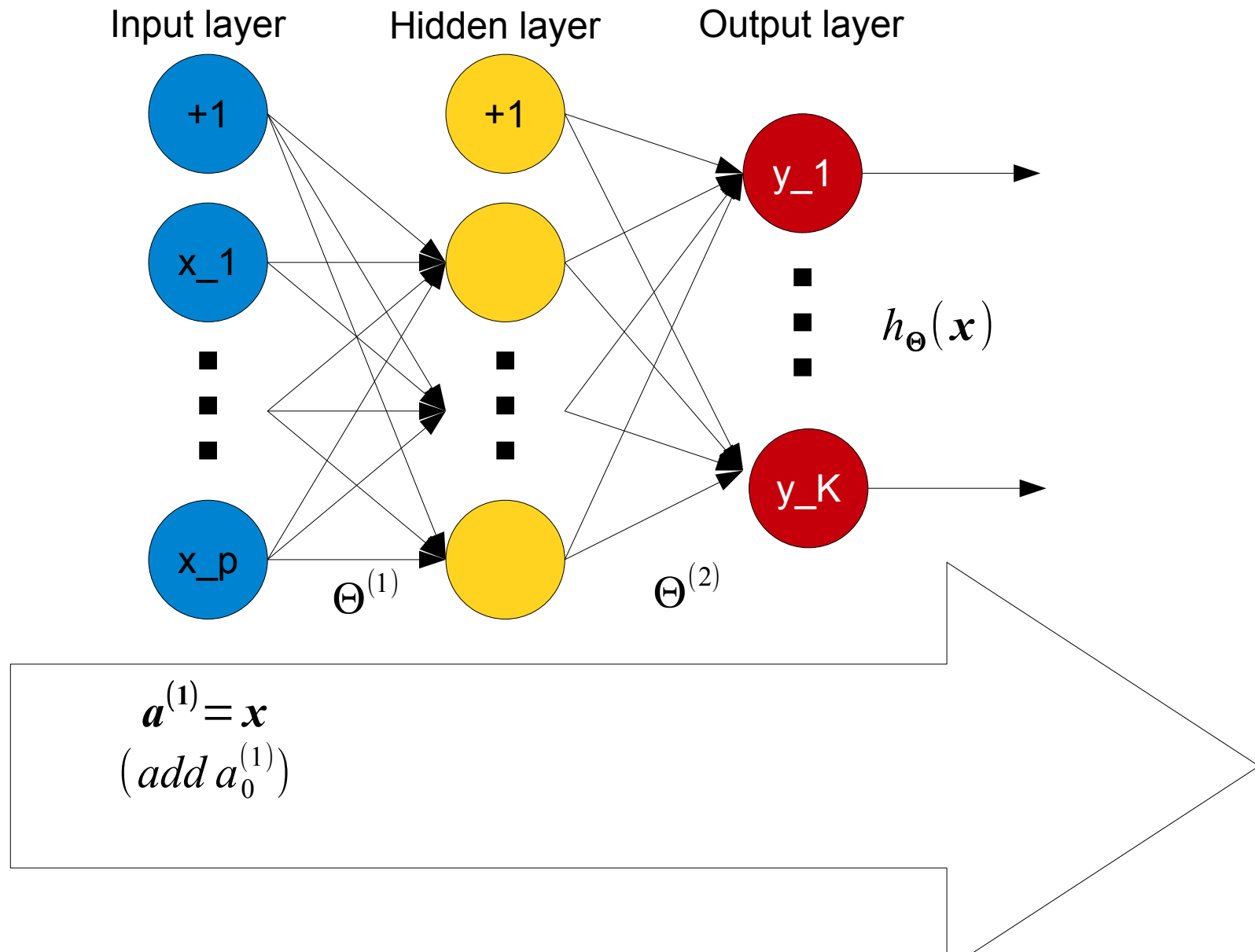
Input layer   Hidden layer   Output layer

+1   +1 y_1

x_1

$h_\Theta(x)$

x_p y_K

$\Theta^{(1)}$ $\Theta^{(2)}$

$$a^{(1)} = x$$
$$\left(add\ a_0^{(1)} \right)$$

Feedforward Propagation

Input layer   Hidden layer   Output layer

$+1$

$a^{(1)}$

$+1$

y_1

$a^{(1)}$

y_K

$h_{\Theta}(x)$

$\Theta^{(1)}$ $\Theta^{(2)}$

$a^{(1)} = x$

$\left(add\ a_0^{(1)} \right)$

Feedforward Propagation

Input layer   Hidden layer   Output layer

$+1$ $+1$

$a^{(1)}$ $z^{(2)}$ y_1

$h_\Theta(x)$

$a^{(1)}$ $z^{(2)}$ y_K

$\Theta^{(1)}$ $\Theta^{(2)}$

$$a^{(1)} = x \qquad z^{(2)} = \Theta^{(1)} a^{(1)}$$
$$\left(add\ a_0^{(1)} \right)$$

Feedforward Propagation

Input layer   Hidden layer   Output layer

$$\boldsymbol{a^{(1)}} = \boldsymbol{x}$$
$$\left(add\ a_0^{(1)} \right)$$

$$\boldsymbol{z^{(2)}} = \Theta^{(1)} \boldsymbol{a^{(1)}}$$
$$\boldsymbol{a^{(2)}} = g\left(\boldsymbol{z^{(2)}} \right)$$
$$\left(add\ a_0^{(2)} \right)$$

Feedforward Propagation

Input layer   Hidden layer   Output layer

$$a^{(1)} = x$$
$$(add\ a_0^{(1)})$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$
$$a^{(2)} = g(z^{(2)})$$
$$(add\ a_0^{(2)})$$

$$z^{(3)} = \Theta^{(2)} a^{(2)}$$

Feedforward Propagation

Input layer   Hidden layer   Output layer

$$a^{(1)} = x$$
$$(add \; a_0^{(1)})$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$
$$a^{(2)} = g(z^{(2)})$$
$$(add \; a_0^{(2)})$$

$$z^{(3)} = \Theta^{(2)} a^{(2)}$$
$$a^{(3)} = g(z^{(3)}) = h_\Theta(x)$$
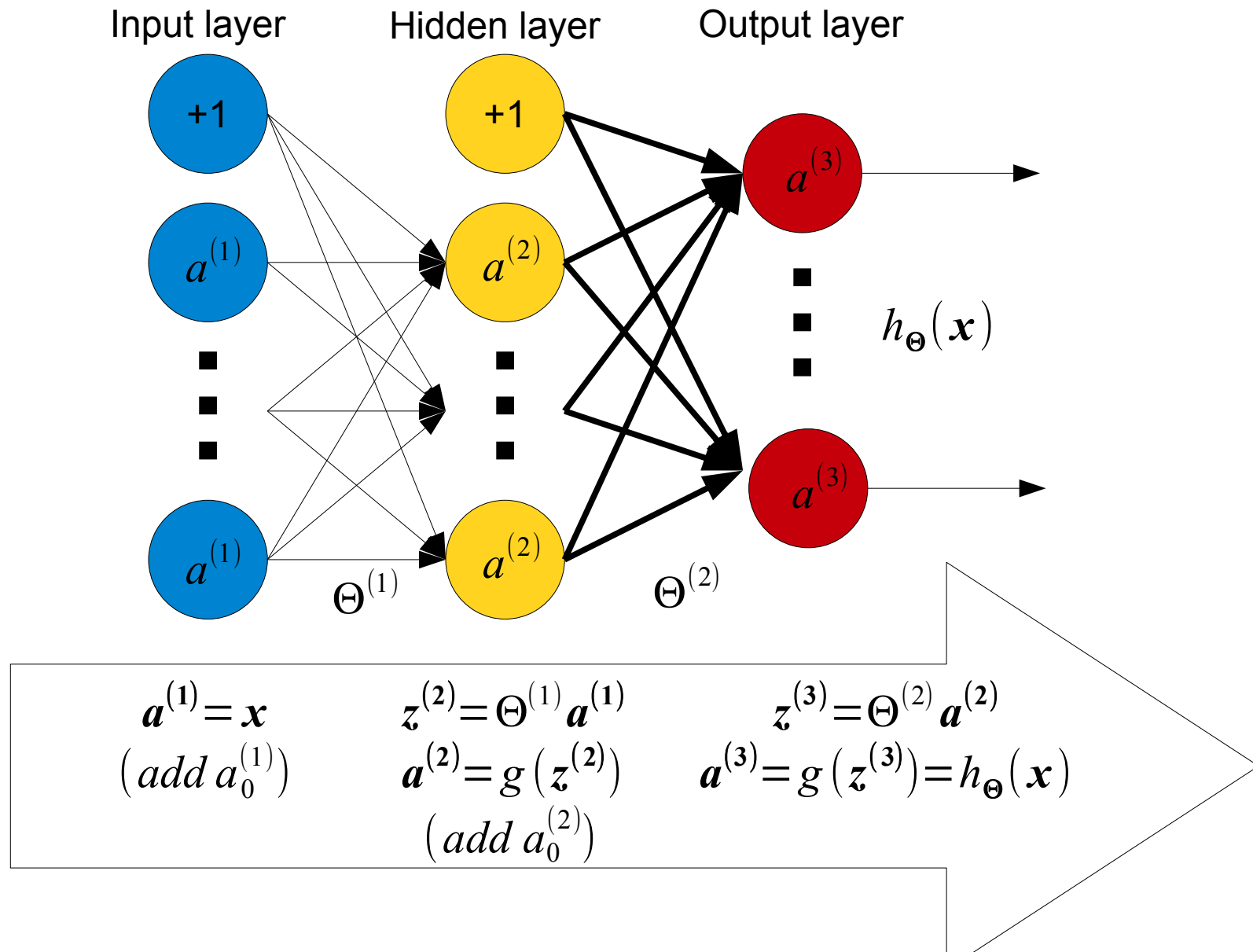
Algorithm: Feedforward Propagation

- Input $\{(x_1),\dots,(x_m)\}, \Theta_{ij}^{(l)}$

- Output $\{h_\Theta(x_1),\dots,h_\Theta(x_m)\}$

- Procedure

  - Set $a^{(1)}=x$

  - For i=2:L

$$z^{(l)}=\Theta^{(l-1)}a^{(l-1)}$$
$$a^{(l)}=g\left(z^{(l)}\right)$$

  - $h_\Theta(x)=a^{(L)}$

Ex 1: Feedforward propagation (Prediction)

- Load data and variables by
  - load('MNIST.mat')
  - load('Weights.mat');
- Complete the following function
  - Function pred = feedforward(Theta1, Theta2, X)
    - where pred is 5000 x 10 matrix of prediction results
- If implemented correctly,
  - [M I] = max(pred, [], 2)
    - Returns 0...,0, 1...,1, 2...,2, 3...,3 …
- The code is prepared by Andrew Ing.
  - https://www.coursera.org/learn/machine-learning/

Dataset

- MNIST dataset

  - X: 5000 x 256 (digits 0-9 from 500 people. Each image consists of 16 x 16 pixels)

    - Visualization: `> imagesc(reshape(X(1,:),[16 16]))`

  - Xy: 5000 x 1 true digits

  - Theta1': 257 x 25 learnt parameters for the first layer.

  - Theta2': 26 x 10 learnt parameters for the second layer.

  - Tips: you need to augment matrices with a column of ones from left; X = [ones(5000,1) X]

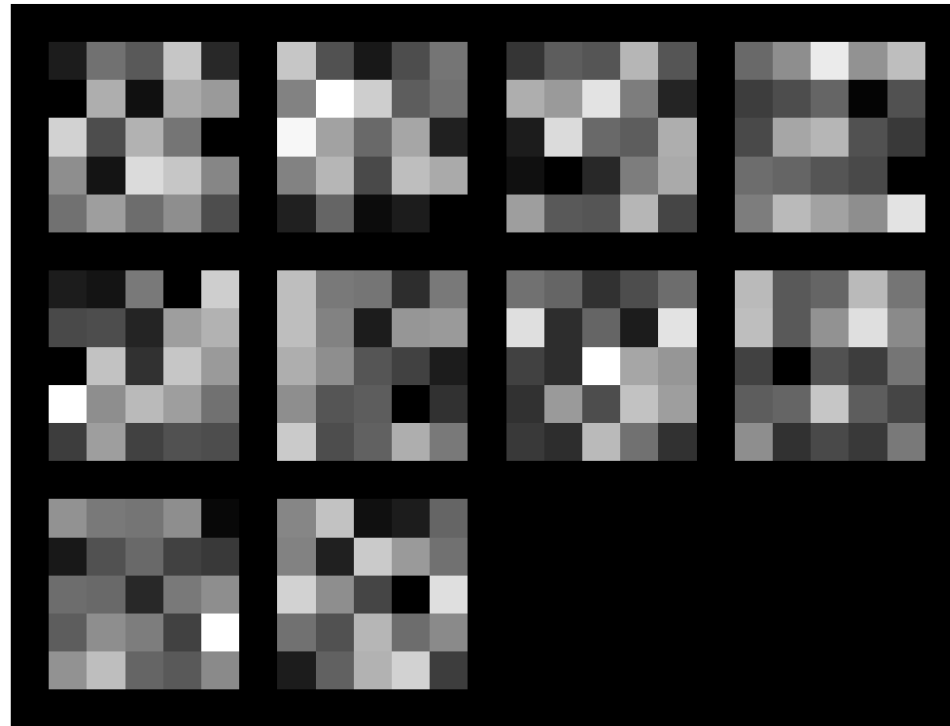 - Tips; you need a "sigmoid" function, or h.m from the previous exercise.

Theta1: 256 x 25

Theta1
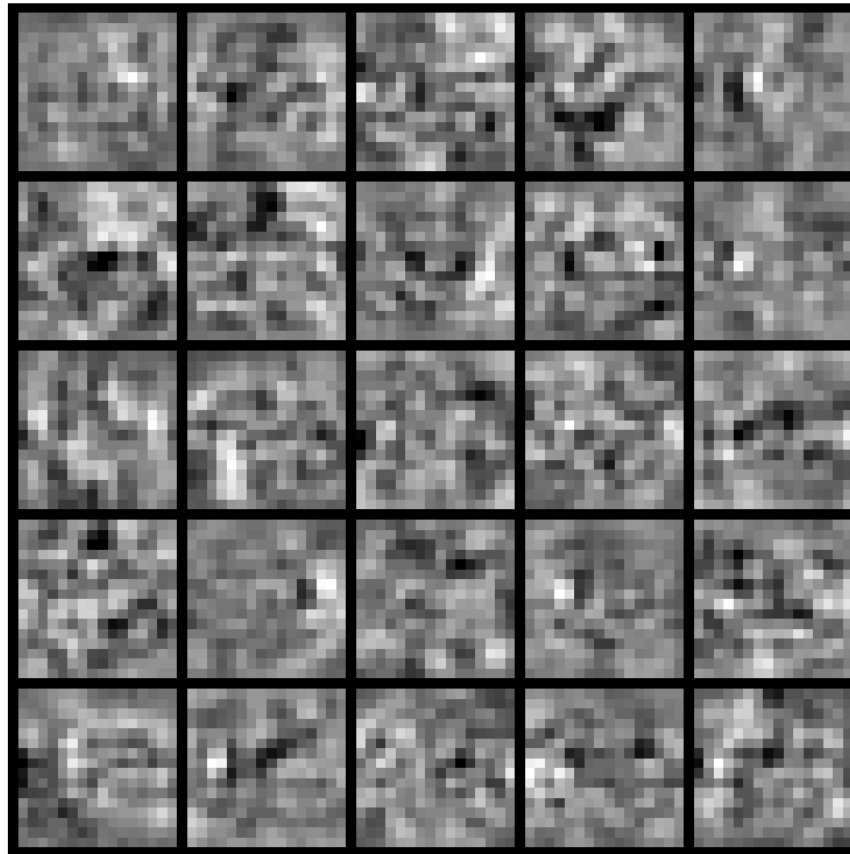
Theta2: 25 x 10

Theta2

Overall structure

Input layer
n x 16 x 16

digit samples

Hidden layer 1
16 x 16 x 5 x 5

Theta1

Hidden layer 2
5 x 5 x 10

Theta2

Output layer
n x 10

1
0
0
0
0
0
0
0
0
0
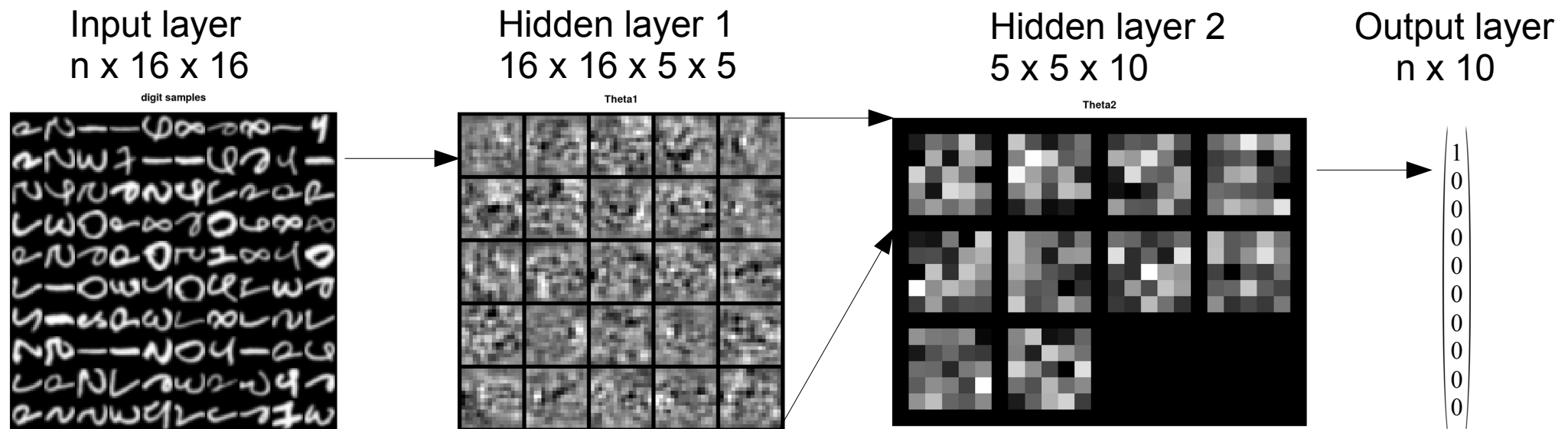
Objective function

- L2-Logistic regression

$$y_i = \{0, 1\}, h(.) \in R$$

$$J(\boldsymbol{\theta}) = -\sum\nolimits_{i=1}^{n} \{y_i \log h(\boldsymbol{\theta}' \boldsymbol{x_i}) + (1-y_i) \log(1 - h(\boldsymbol{\theta}' \boldsymbol{x_i}))\} + \frac{\lambda}{2} \sum\nolimits_{j=1}^{p} \theta_j^2$$

- Multi-output neural nets for classification

$$y_i^{(k)} = \{0, 1\}^k, h_k \in R^k$$

$$J(\boldsymbol{\Theta}) = -\sum\nolimits_{i=1}^{n} \sum\nolimits_{k=1}^{K} \{y_i^{(k)} \log h_{\boldsymbol{\Theta}}(\boldsymbol{x_i})^k + (1 - y_i^{(k)}) \log(1 - h_{\boldsymbol{\Theta}}(\boldsymbol{x_i})^k)\}$$

$$+ \frac{\lambda}{2} \sum\nolimits_{l=1}^{L-1} \sum\nolimits_{i=1}^{n} \sum\nolimits_{j=1}^{p} (\Theta_{ij}^{(l)})^2$$

Gradient of the objective

- Gradient of the objective w.r.t. parameters $\dfrac{\partial J(\Theta)}{\partial \Theta_{ji}^{(l)}}$

  - using a chain rule $\dfrac{\partial J(\Theta)}{\partial \Theta_{ji}^{(l)}} = \dfrac{\partial J(\Theta)}{\partial z_i^{(l)}} \dfrac{\partial z_i^{(l)}}{\partial \Theta_{ji}^{(l)}}$

  - Let $\delta_i^{(l+1)} = \dfrac{\partial J(\Theta)}{\partial z_i^{(l)}}$ be an "error" (at node j in the l-th layer) propagated from the (l+1)-th layer

  - Since $z_i^{(l)} = \sum_i \Theta_{ji}^{(l)} a_j^{(l)}, \quad \dfrac{\partial z_i^{(l)}}{\partial \Theta_{ji}^{(l)}} = \dfrac{\partial}{\partial \Theta_{ji}^{(l)}}\left(\sum_i \Theta_{ji}^{(l)} a_j^{(l)} \right) = a_j^{(l)}$

  - Plugging in the above relations obtains

$$\frac{\partial J(\Theta)}{\partial \Theta_{ji}^{(l)}} = \delta_i^{(l+1)} a_j^{(l)}$$

Gradient Computation

- Input: $\{(x_{1,}y_1),\dots,(x_m,y_m)\},\Theta_{ij}^{(l)}$

- Output: $\Delta_{ij}^{(l)}=\dfrac{\partial J(\Theta)}{\partial \Theta_{ij}^{(l)}}=a_j^{(l)}\delta_i^{(l+1)}$

- Procedure

  - For i=1:m (each example) Feedforward propagation

    Set $a^{(1)}=x^{(i)}$

    - For I=1:L Feedforward propagation to compute $a^{(l)}$

    - Compute $\delta^{(L)}=a^{(L)}-y$

    - For I=L:2 Compute $\delta^{(l)}=(\Theta^{(l)})^T\delta^{(l+1)}.*g'(z^{(l)})$

    - Update $\Delta_{ij}^{(l)}=\Delta_{ij}^{(l)}+a_j^{(l)}\delta_i^{(l+1)}$ Backward propagation

Backward Propagation (L=3)

Input layer Hidden layer Output layer

+1 +1 $\delta^{(3)}$

x_1 $h_\Theta(x)$

x_p $\delta^{(3)}$

$\Theta^{(1)}$ $\Theta^{(2)}$
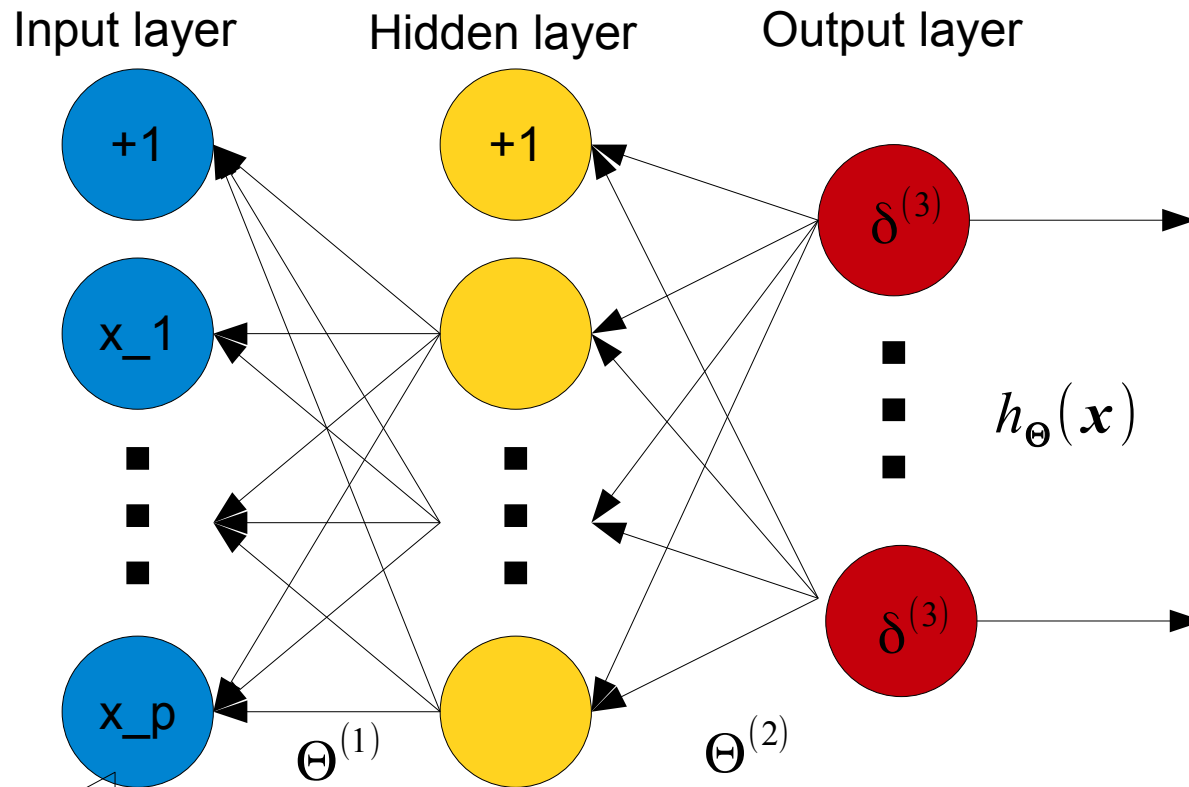
Difference between current prediction and label $\delta^{(3)} = a^{(3)} - y$

Backward Propagation (L=3)

Input layer   Hidden layer   Output layer
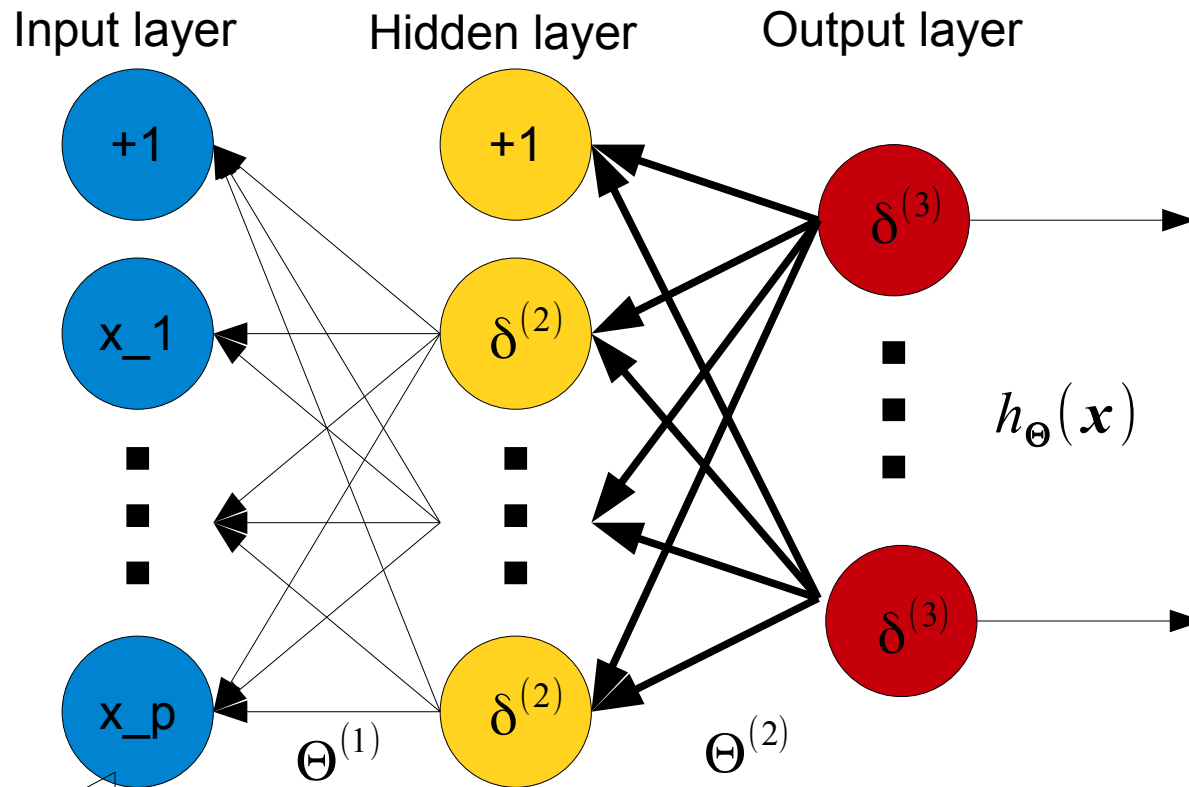
$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)}) \qquad \delta^{(3)} = a^{(3)} - y$$

Backward Propagation (L=3)

Input layer Hidden layer Output layer

$+1$ $+1$ $\delta^{(3)}$

x_1 $\delta^{(2)}$ $h_{\Theta}(x)$

x_p $\delta^{(2)}$ $\delta^{(3)}$

$\Theta^{(1)}$ $\Theta^{(2)}$
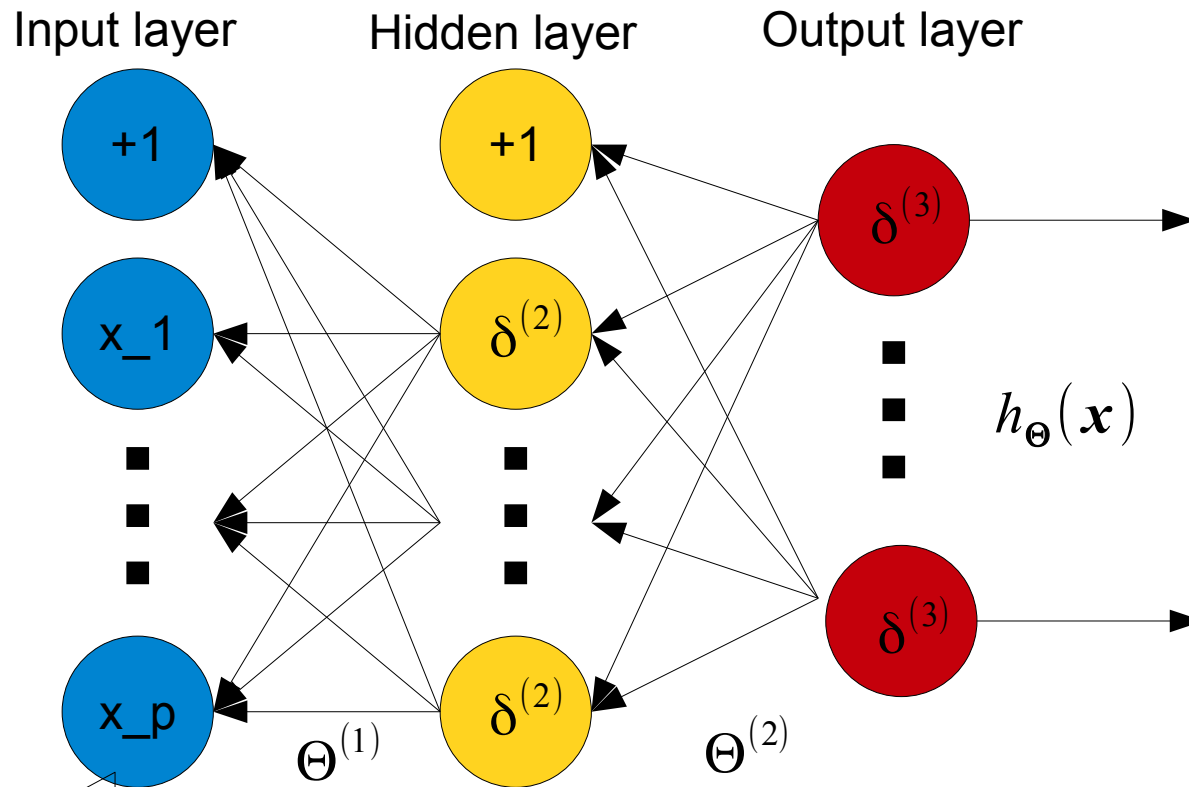
Until the end (beginning).. $\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$ $\delta^{(3)} = a^{(3)} - y$

Understanding the update rule

$$\delta^{(l)}=(\Theta^{(l)})^T \delta^{(l+1)}.*g'(z^{(l)}) \qquad \delta_i^{(l+1)}=\frac{\partial J(\Theta)}{\partial z_i^{(l)}}$$

- We focus on one delta (j), and apply chain rule.

$$\delta_j^{(l)}=\frac{\partial J(\Theta)}{\partial z_j^{(l)}}=\sum_k \frac{\partial J(\Theta)}{\partial z_k^{(l)}}\frac{\partial z_k^{(l)}}{\partial z_j^{(l)}}=\sum_k \delta_k^{(l+1)}\frac{\partial z_k^{(l)}}{\partial z_j^{(l)}}$$

- Since $z_k^{(l)}=\sum_j \Theta_{kj}^{(l)} a_j^{(l)}$ and $a_j^{(l)}=g(z_j^{(l)})$,

$$\frac{\partial z_k^{(l)}}{\partial z_j^{(l)}}=\frac{\partial}{\partial z_j^{(l)}}\sum_j \Theta_{kj}^{(l)} a_j^{(l)}=\frac{\partial}{\partial z_j^{(l)}}\sum_j \Theta_{kj}^{(l)} g(z_j^{(l)})=\Theta_{kj}^{(l)} g'(z_j^{(l)})$$

- Plugging in the above relationships obtains

$$\delta_j^{(l)}=\sum_k \delta_k^{(l+1)}\frac{\partial z_k^{(l)}}{\partial z_j^{(l)}}=g'(z_j^{(l)})\sum_k \delta_k^{(l+1)}\Theta_{kj}^{(l)}=(\Theta^{(l)})^T \delta^{(l+1)}.*g'(z_j^{(l)})$$

Gradient Computation

- Input: $\{(x_1, y_1), \ldots, (x_m, y_m)\}, \Theta_{ij}^{(l)}$

- Output: $\Delta_{ij}^{(l)} = \dfrac{\partial J(\Theta)}{\partial \Theta_{ij}^{(l)}} = a_j^{(l)} \delta_i^{(l+1)}$

- Procedure

  - For i=1:m (each example) Feedforward propagation

    Set $a^{(1)} = x^{(i)}$

    – For I=1:L Feedforward propagation to compute $a^{(l)}$

    – Compute $\delta^{(L)} = a^{(L)} - y$

    – For I=L:2 Compute $\delta^{(l)} = (\Theta^{(l)})^T \delta^{(l+1)} .* g'(z^{(l)})$

    – Update $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ Backward propagation

Optimization by gradient descent

$$\Theta_{ij}^{(l)} = \Theta_{ij}^{(l)} + \epsilon \, \Delta_{ij}^{(l)}$$

- Once gradient is computed, then variants of gradient descent methods are available
  - SGD, Momentum, AdaGrad, Adam

- Newton's method is too expensive.

- Only local convergence is guaranteed.

Ex 2: neural net training

- Try "nn_demo.m"

> nn_demo

- "nnCostFunction.m" performs gradient computation. Read and compare the code with lecture slides.

- Try different # of iterations, lambda by editing nn_demo.m

  - Can you achieve 100% accuracy ?

Overall structure

| Input layer
n x 16 x 16 | Hidden layer 1
16 x 16 x 5 x 5 | Hidden layer 2
5 x 5 x 10 | Output layer
n x 10 |
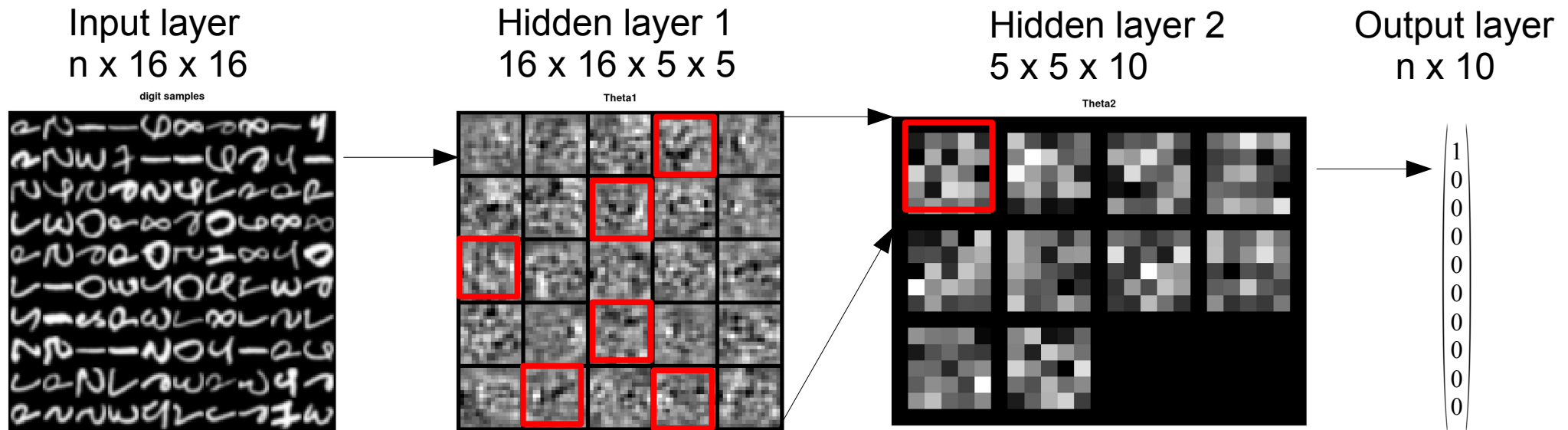
Objective computation in nnCostfunction.m

```matlab
p_vec = zeros(size(X, 1), 1); % prediction vector
for i=1:m

  h1 = sigmoid([1 X(i,:)] * Theta1');
  h2 = sigmoid([1 h1] * Theta2');
  p_vec = h2';


  y_vec = (1:num_labels == y(i))'; % one hot encoding for y, such as y(2) -> [ 0 1 0 0 0 ]


  J += -( y_vec'*log(p_vec) + (1-y_vec)'*log(1-p_vec) ); % objective function
end


J = J/m + lambda/(2*m)*( norm(Theta1(:,2:end)(:)).^2 + norm(Theta2(:,2:end)(:)).^2 ); % +regularization
```

Forward/Backward computation in nnCostfunction.m

```matlab
for t=1:m

% Feedforward
 a_1 = [1; X(t,:)'];
 z_2 = Theta1 * a_1;
 a_2 = [1; sigmoid(z_2)];
 z_3 = Theta2 * a_2;
 a_3 = sigmoid(z_3);

% Backward propagation
 y_vec = (1:num_labels == y(t))'; % label vector in one-hot encoding

 delta_3 = a_3 - y_vec;
 delta_2 = Theta2(:,2:end)' * delta_3 .* sigmoidGradient(z_2);

% Gradient update
 Theta2_grad += delta_3 * a_2';
 Theta1_grad += delta_2 * a_1';

end
```

Appendix

- Index
  - Random initialization
  - Model zoo
  - Activation functions
  - Network design
  - Gradient Checking
  - Software

Random Initialization

- Suppose that weights are initialized by zeros. Then all a and z become zeros as well, and we cannot compute gradients by backprop.

- Thereby weights must be initialized by random values.

- Implemented in "randInitializeWeights.m"

Activation functions

- Logistic sigmoid $\quad g(x) = \dfrac{\exp(-x)}{1 + \exp(-x)}$

- Tanh $\quad g(x) = \tanh(x)$

- Relu $\quad g(x) = x > 0 \, ? \, x : 0$
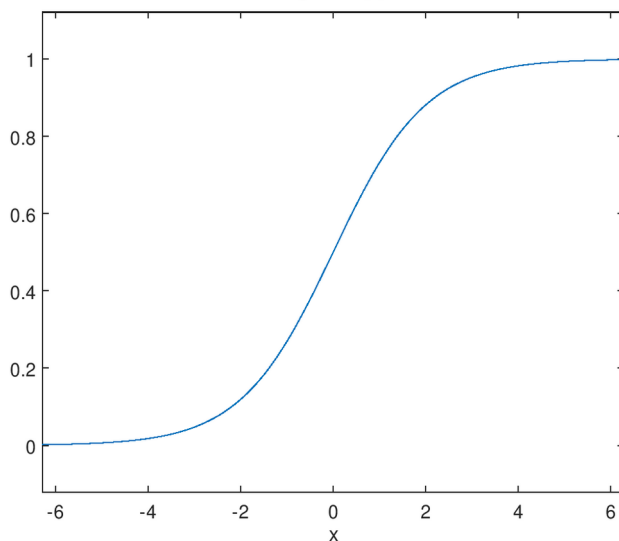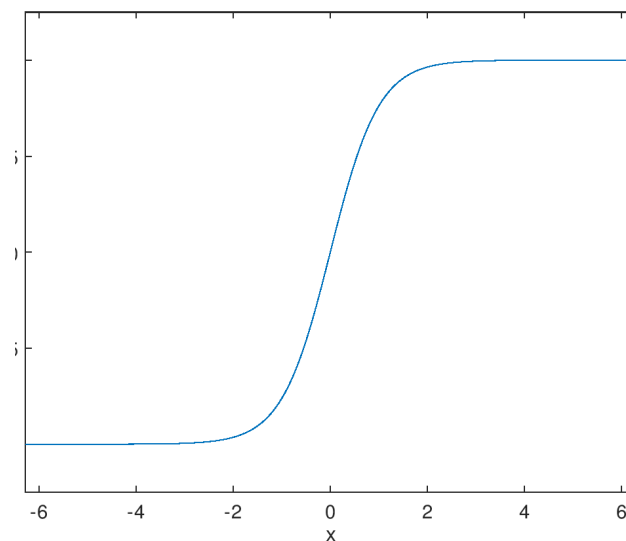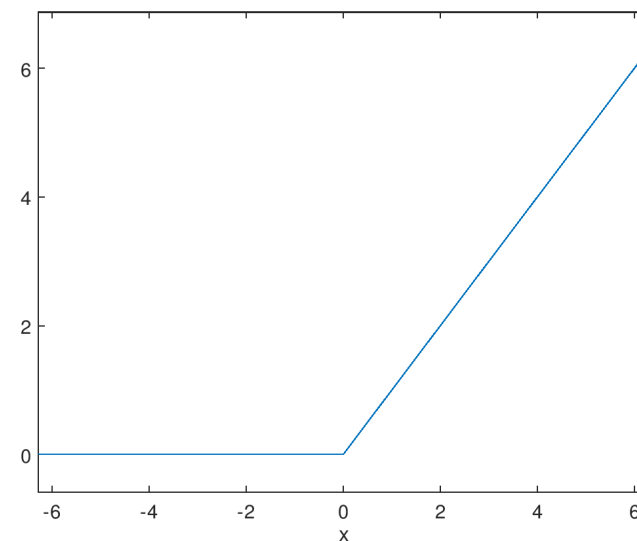
Network design tips

- Number of input units is the same as the number of features.

- Number of output units is the same as the number of classes.

- Number of hidden units is approximately the number of features.

- More number of layers can represent more complex decision boundary.

Multi-layer perceptron (MLP)
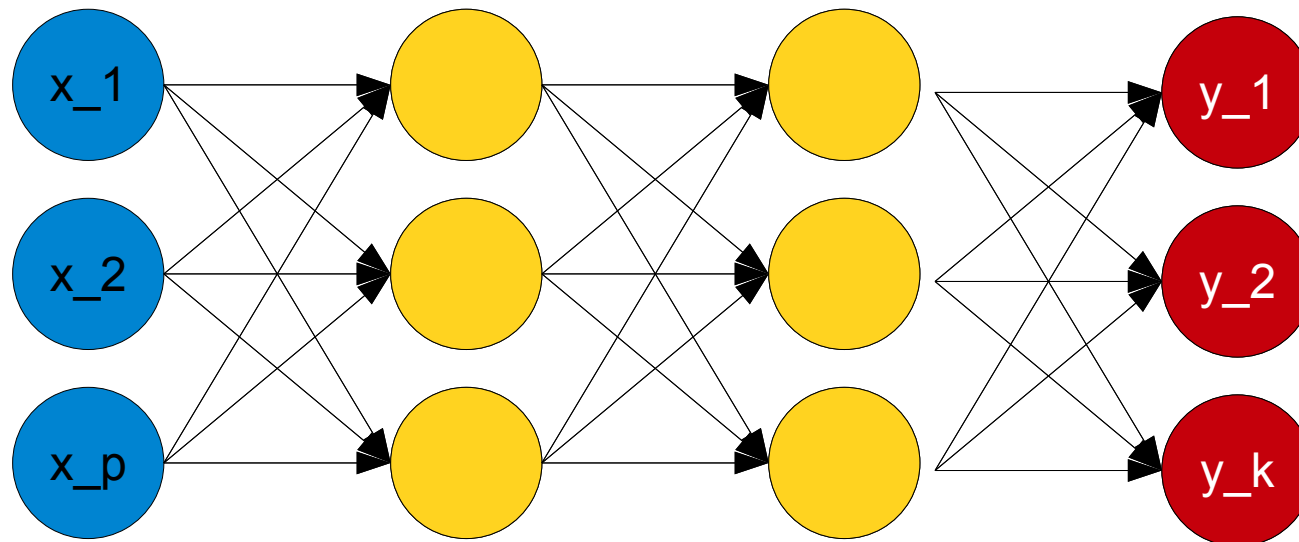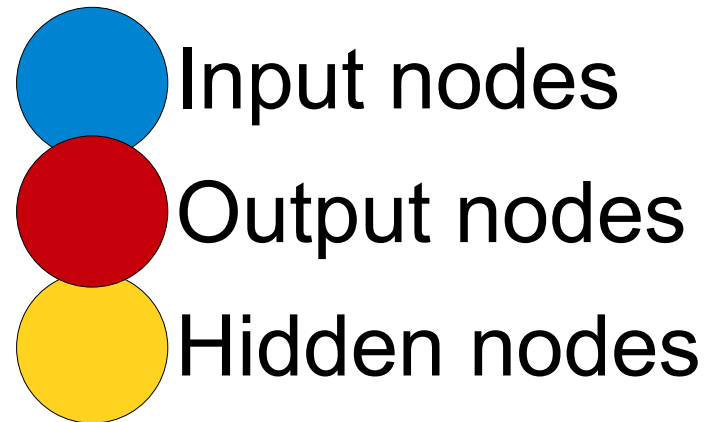
Convolution Neural Network (CNN)
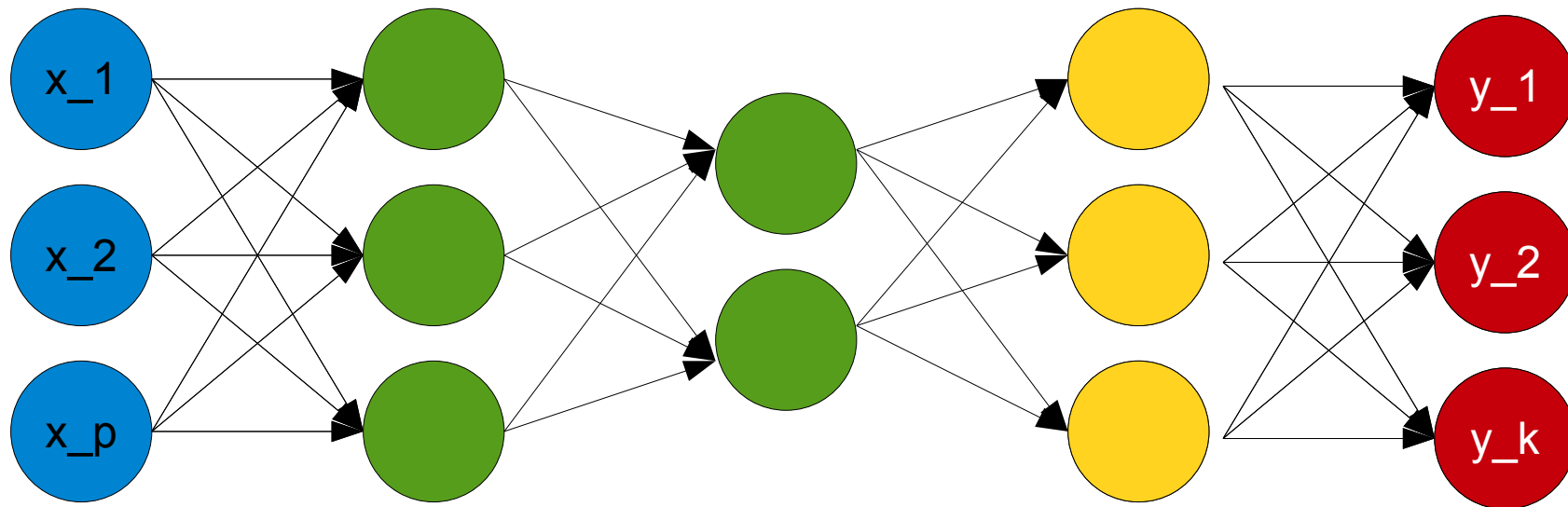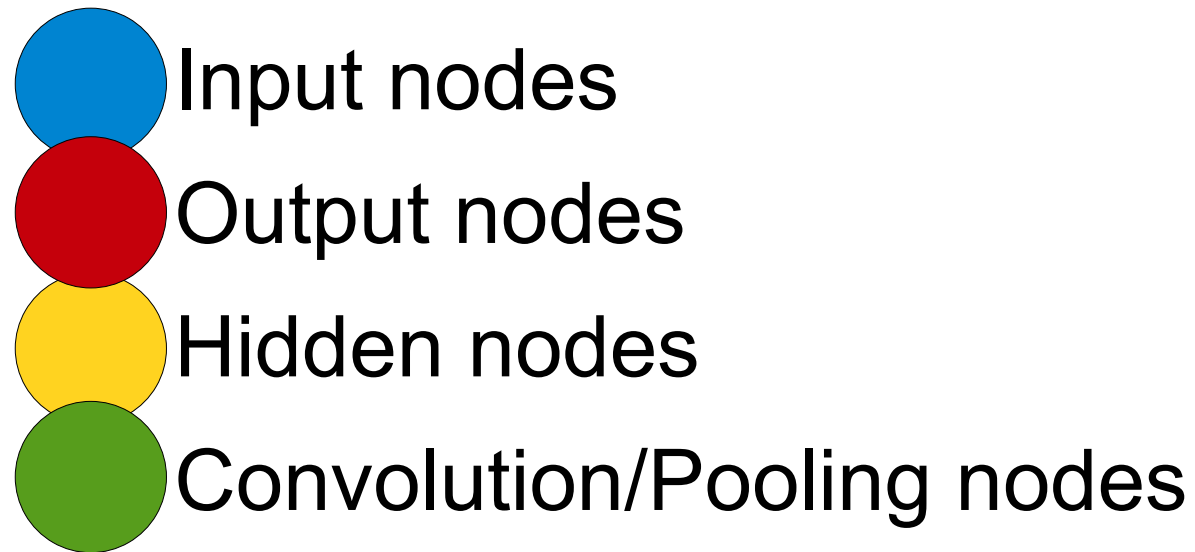
CNN: 1D, 2D and 3D
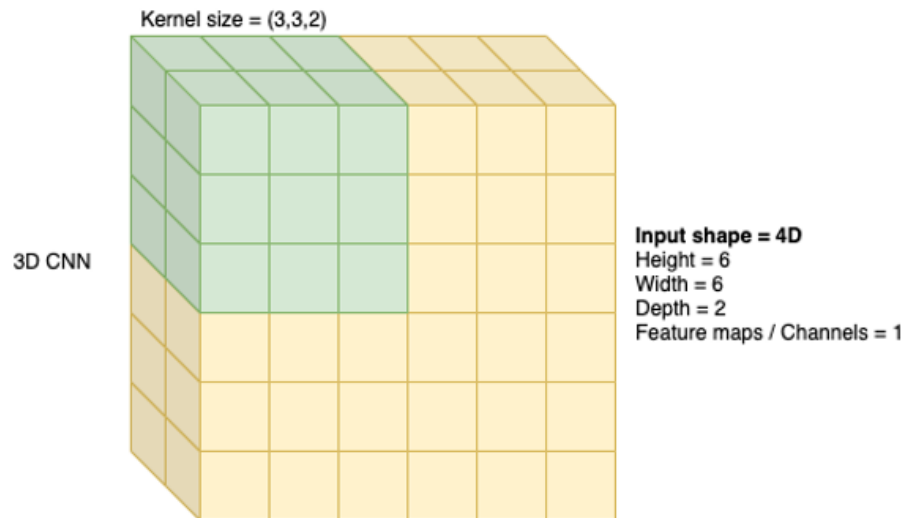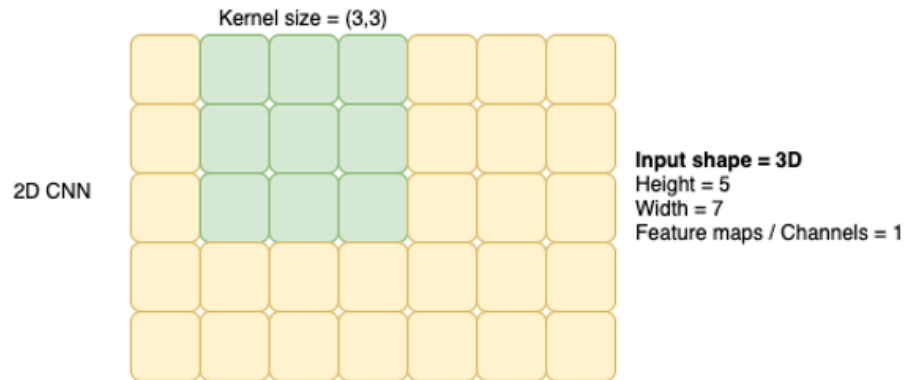
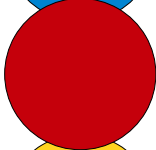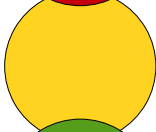These images only have a single channel here,
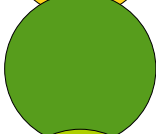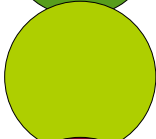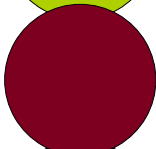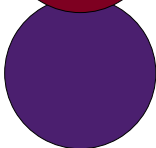but can have multiple such as R, G, B
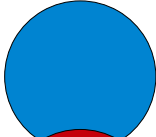
Rucurrent Neural Network (RNN)
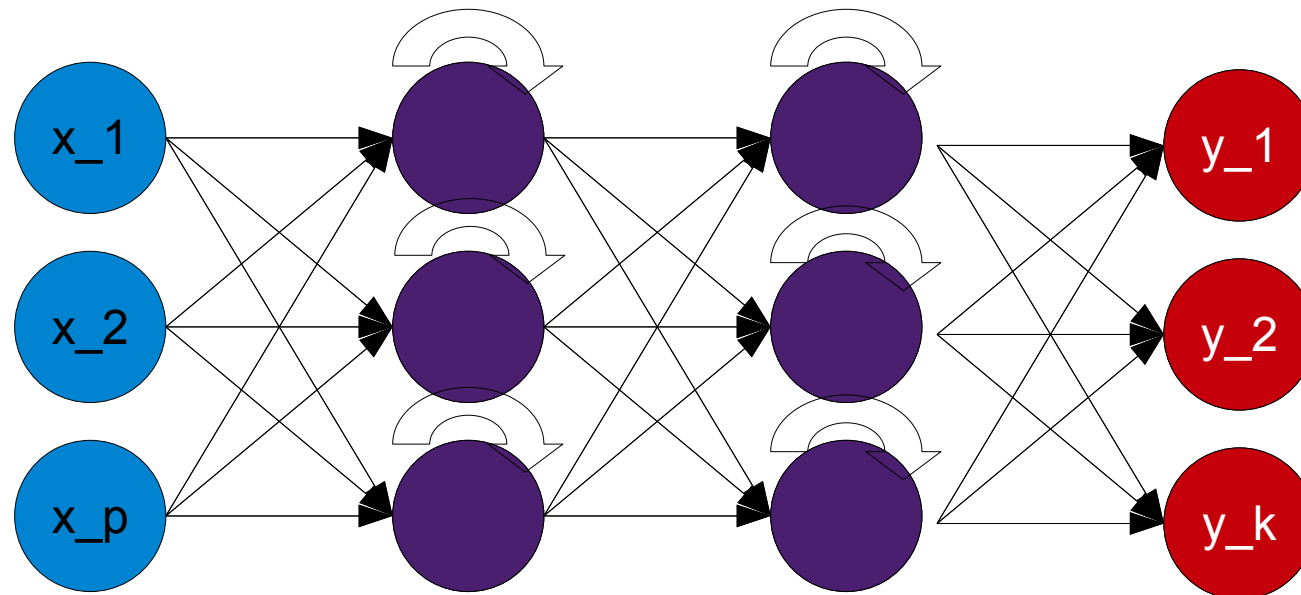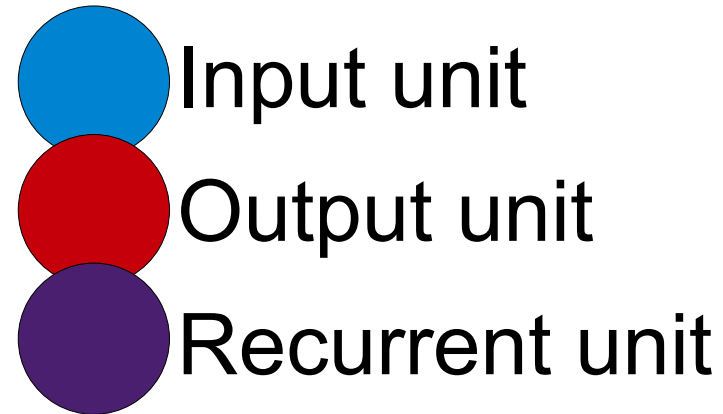
Long/Short Term Memory (LSTM)
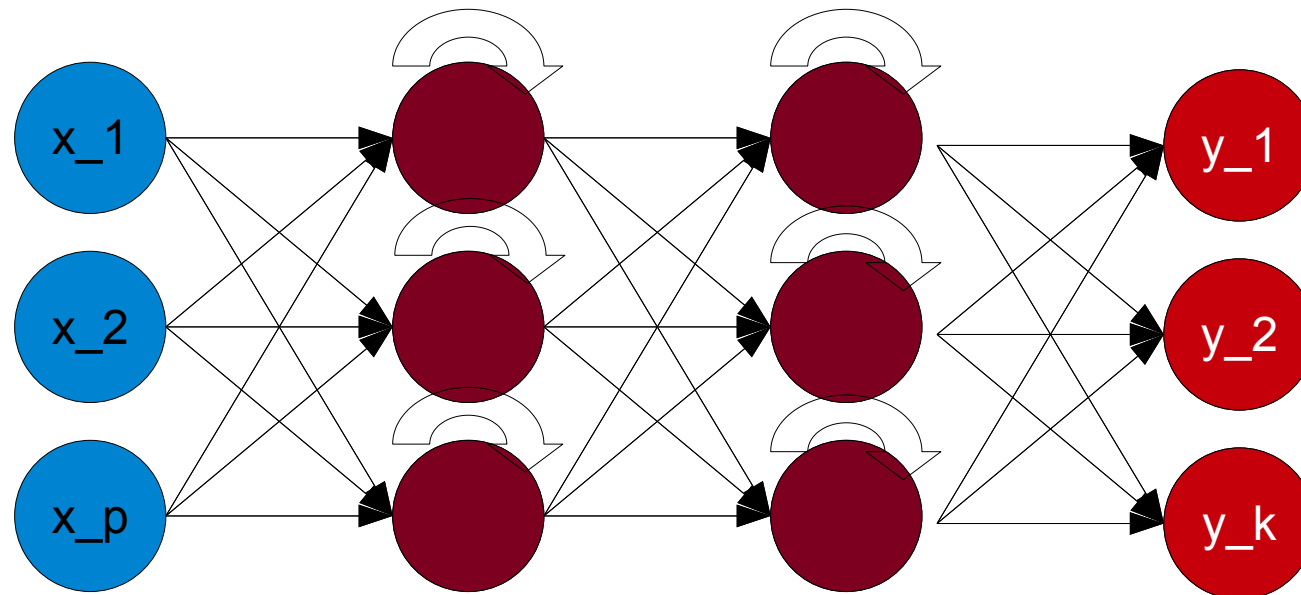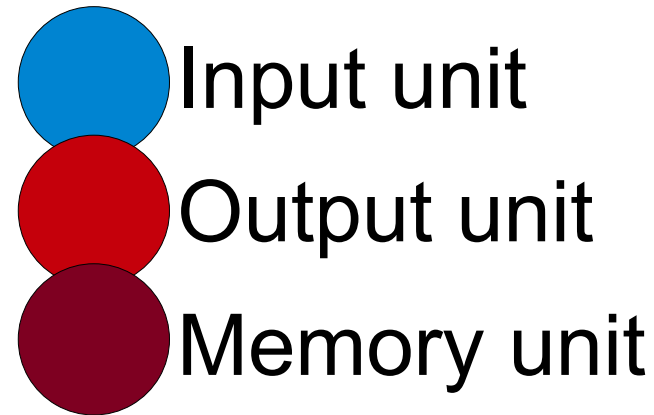
Types of nodes

- 🔵 Input nodes
- 🔴 Output nodes
- 🟡 Hidden nodes
- 🟢 Convolution nodes
- 🟢 Pooling nodes
- 🔴 Memory nodes
- 🟣 Recurrent nodes

Software

- For handling large dataset, use of optimization packages with GPU support is highly recommended.

- Popular packages

    - TensorFlow + Keras (C++/Python)
        - Google
    - Pytorch (C++/Python)
        - Facebook

Methods Covered

- Classification
  - NB, LDA, LogReg, kNN, SVM, NN
- Regression
  - OLS, Lasso, Ridge, GP
- Clustering
  - k-means, EM
- Dimensionality Reduction
  - PCA, CCA