Introduction to Deep Learning

Neural Networks and Deep Learning

Ira A. Fulton Schools of Engineering
Arizona State University

Objective

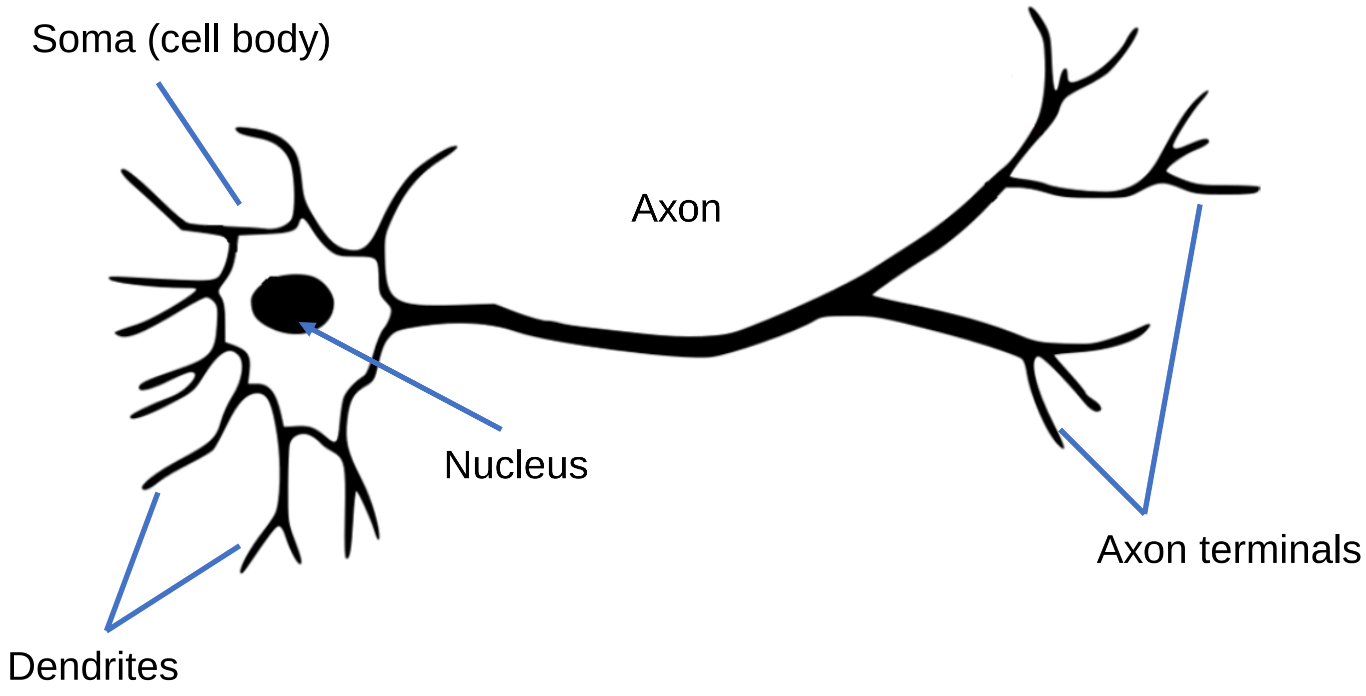
Objective

Describe the big-picture view of how neural networks work

Objective

Identify the basic building blocks and notations of deep neural networks

Illustrating A Biological Neuron



Neural Network

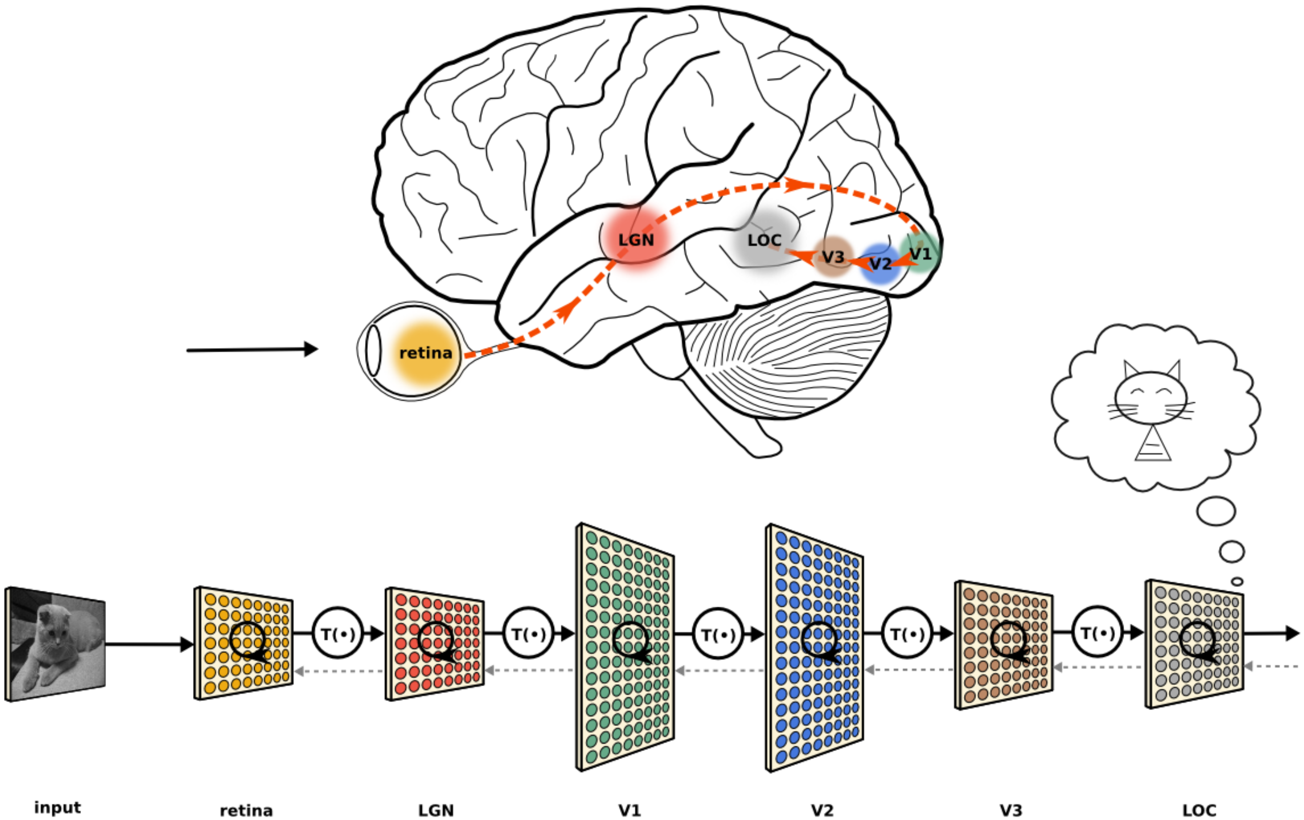
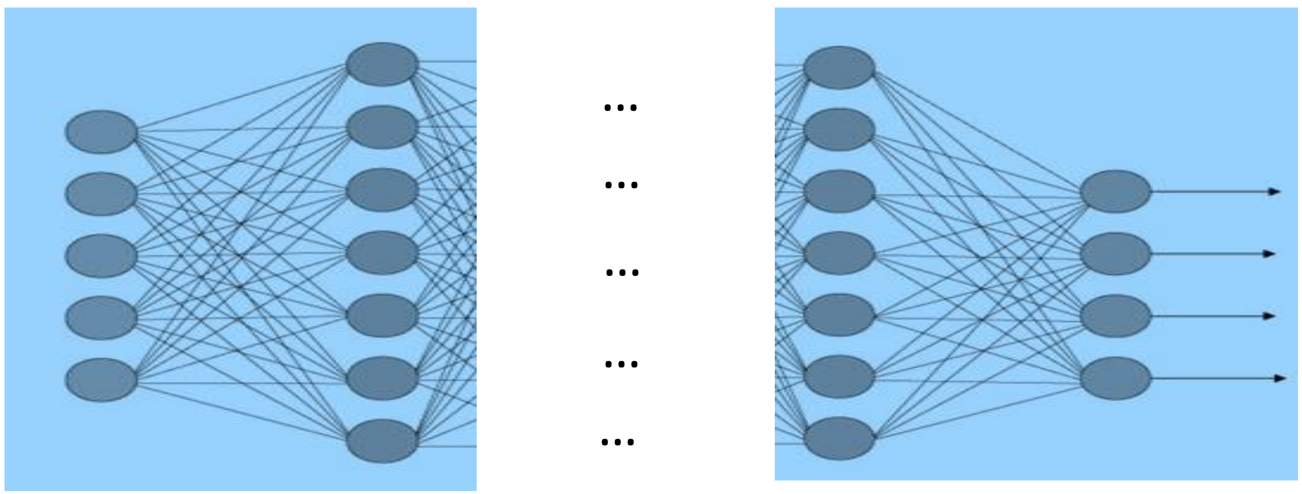
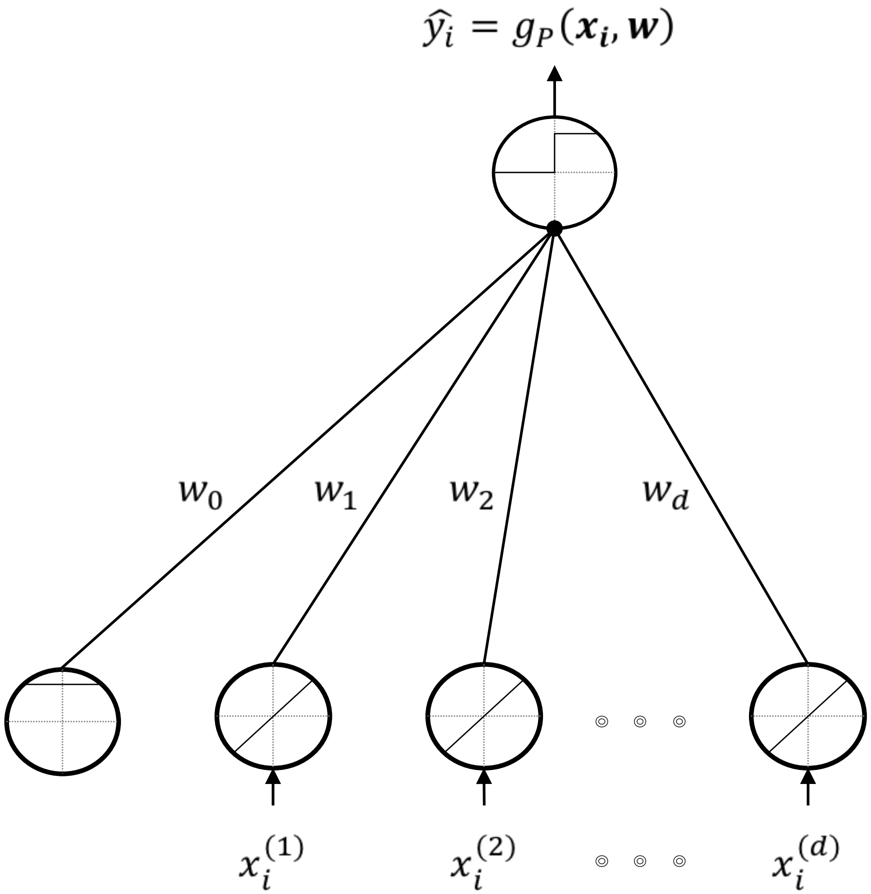


Figure source: https://neuwritesd.org/2015/10/22/deep-neural-networks-help-us-read-your-mind/

Artificial Neural Networks



Building Artificial Neural Networks



Output layer

Activation

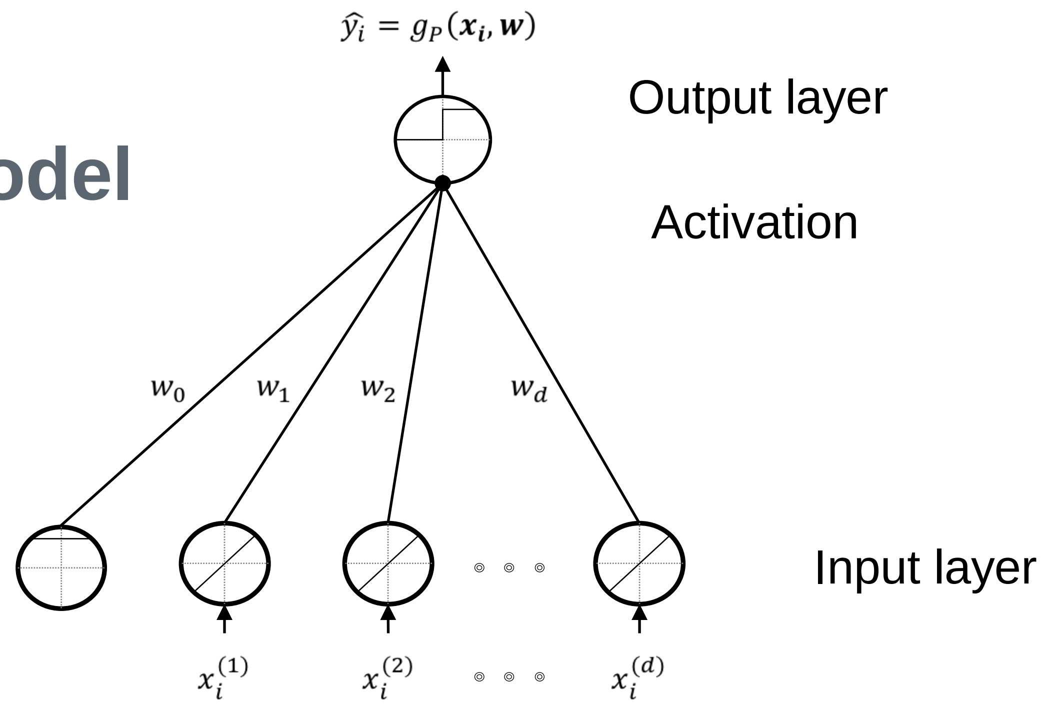
Input layer

Building Artificial Neural Networks (cont'd)

What does this "neuron" do?

$$g_P(\boldsymbol{x}_i, \boldsymbol{w}) = \begin{cases} 1, & \textit{if } \boldsymbol{w}^T\boldsymbol{x}_i > 0 \\ 0, & \text{otherwise} \end{cases}$$

The Perceptron model



Logistic Neuron

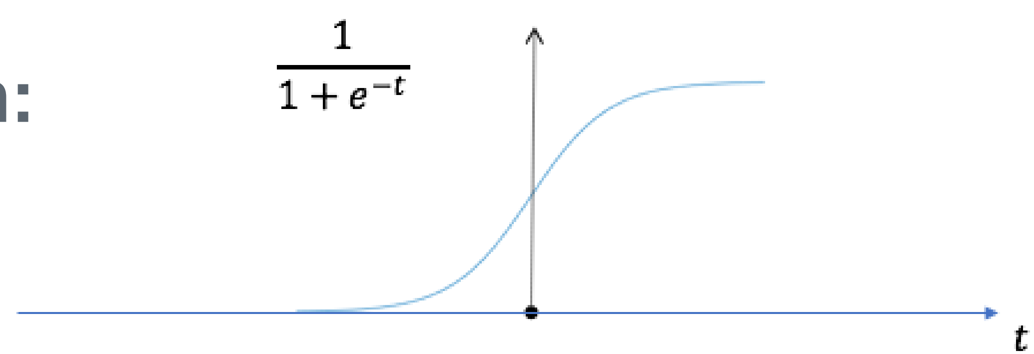
In Perceptron:

$$g_P(\boldsymbol{x_i}, \boldsymbol{w}) = \begin{cases} 1, & \text{if } \boldsymbol{w^T x_i} > 0 \\ 0, & \text{otherwise} \end{cases}$$

If we let:

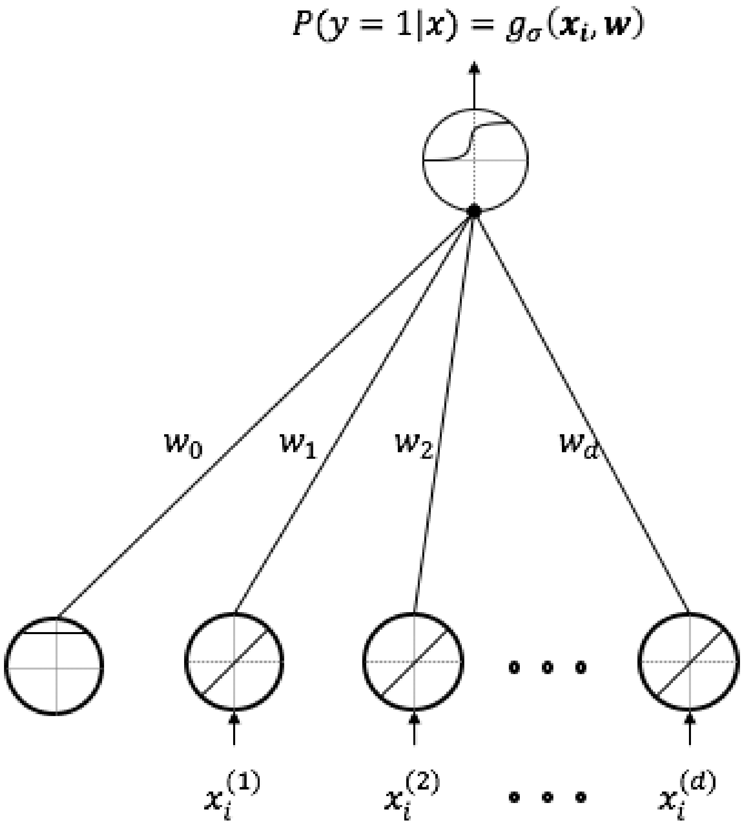
$$g_\sigma(\boldsymbol{x_i}, \boldsymbol{w}) = \frac{1}{1 + e^{-\boldsymbol{w}^T \boldsymbol{x}_i}}.$$

The logistic function:



Logistic Neuron (cont'd)

We have:



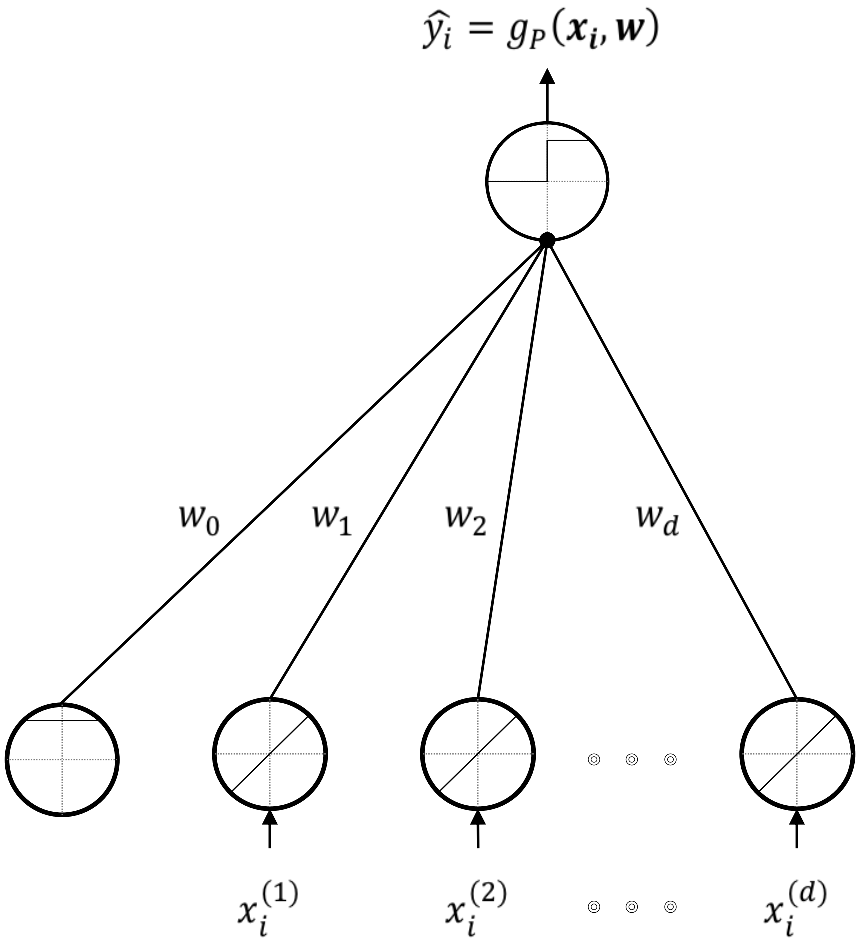
A probability prediction

Activation

Input layer

Learning in the Perceptron

- "Learning": how does the neuron adapt its weights in response to the inputs?



The Perceptron Learning Algorithm

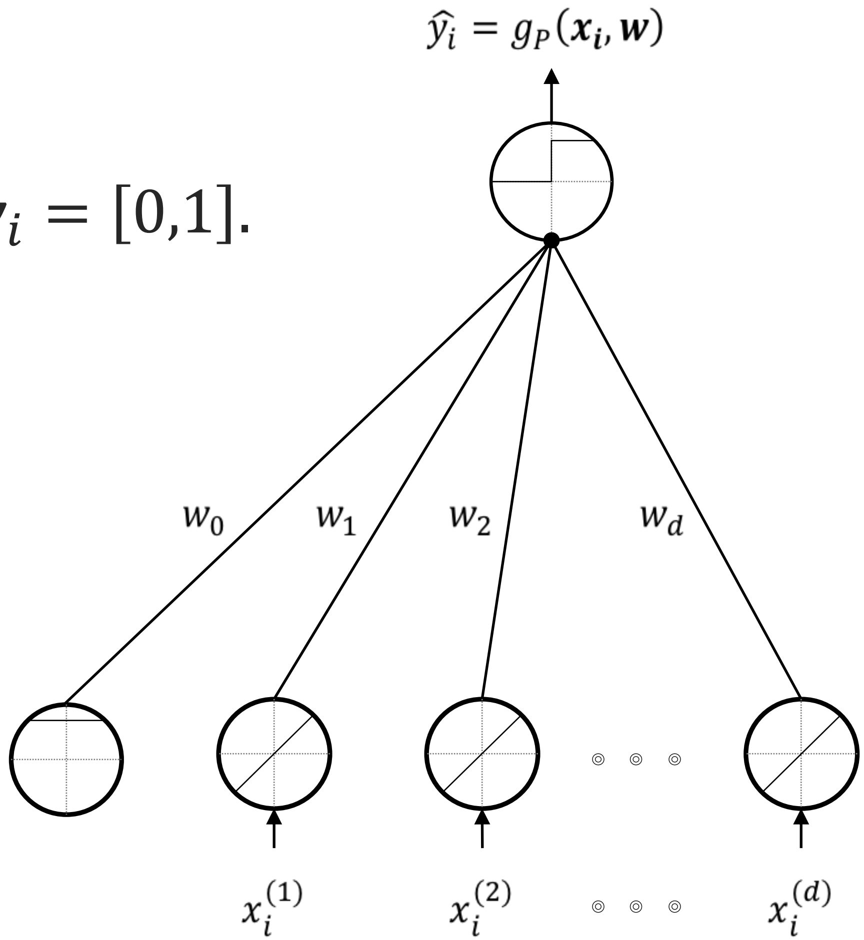
Input

- Training set

$D = \{(x_i, y_i), i \in [1,2, \dots n]\}. y_i = [0,1].$

Initialization

- Initialize the weights w(0) (and some thresholds)
- Weights may be set to 0 or small random values



The Perceptron Learning Algorithm (cont'd)

Iterate for t until a stop criterion is met

{

for each sample x_i with label y_i:

{

compute the output $\check{y}_i$ of the network

estimate the error of the network $e\big(w(t)\big) = \ y_i - \check{y}_i$

update the weight $w(t+1) = w(t) + e\big(w(t)\big)x_i$

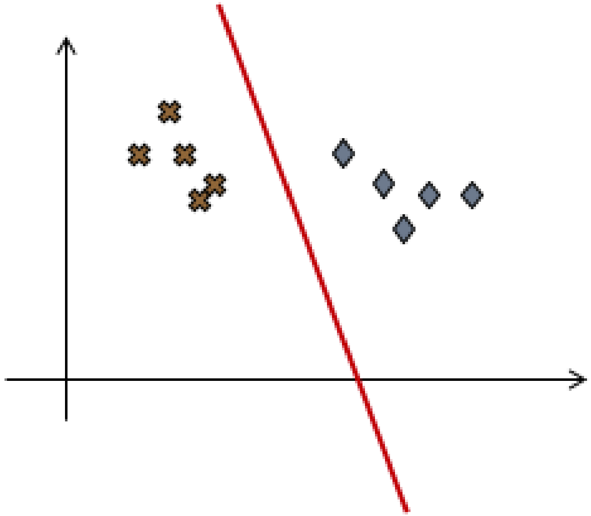
}

t++

}

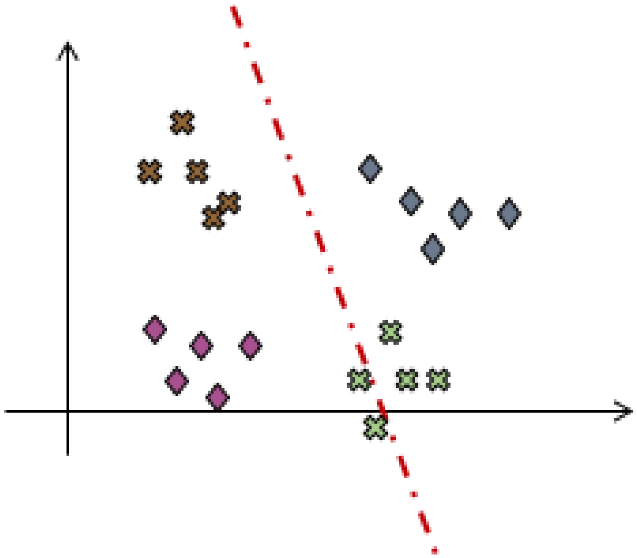
The Need for Multiple Layers

- This is easy and can be learned by the Perceptron Algorithm



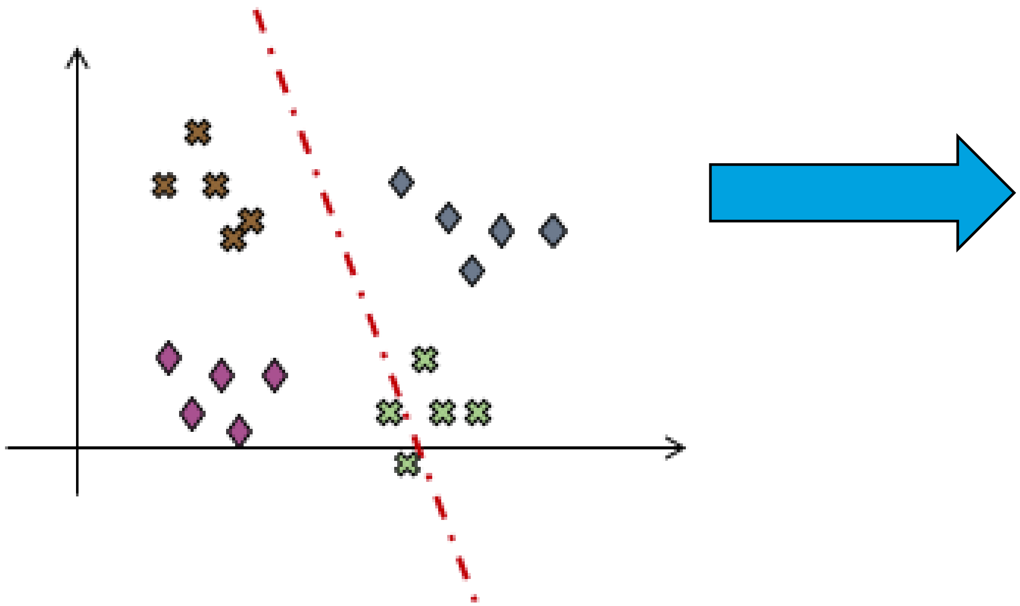
$$w_1x_1 + w_2x_2 + w_0 = 0$$

- But how about this?



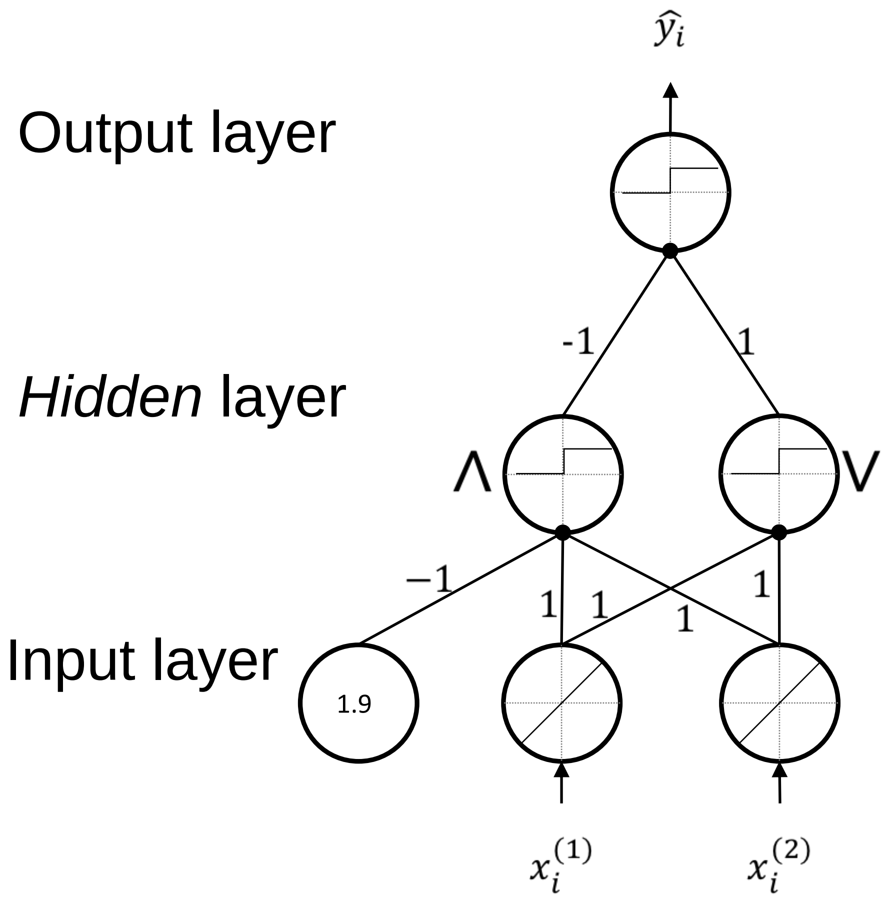
Extending to Multi-layer Neural Networks

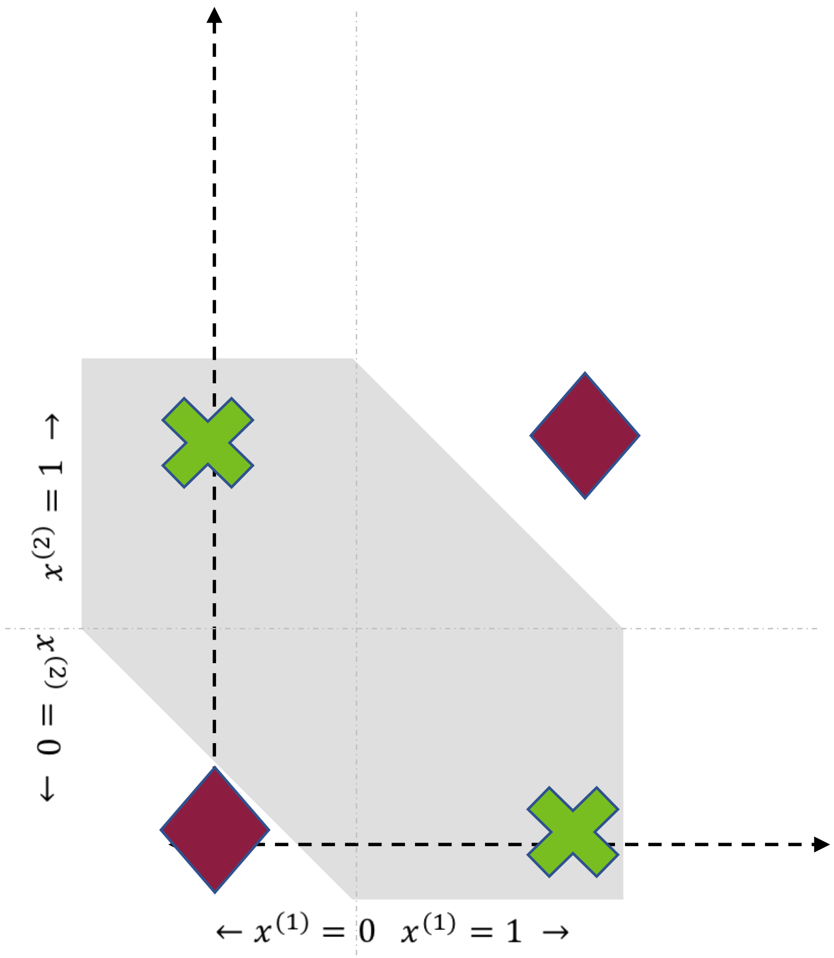
Question: Can a multi-layer version of the Perceptron (MLP) help solving the XOR problem?



The XOR Problem

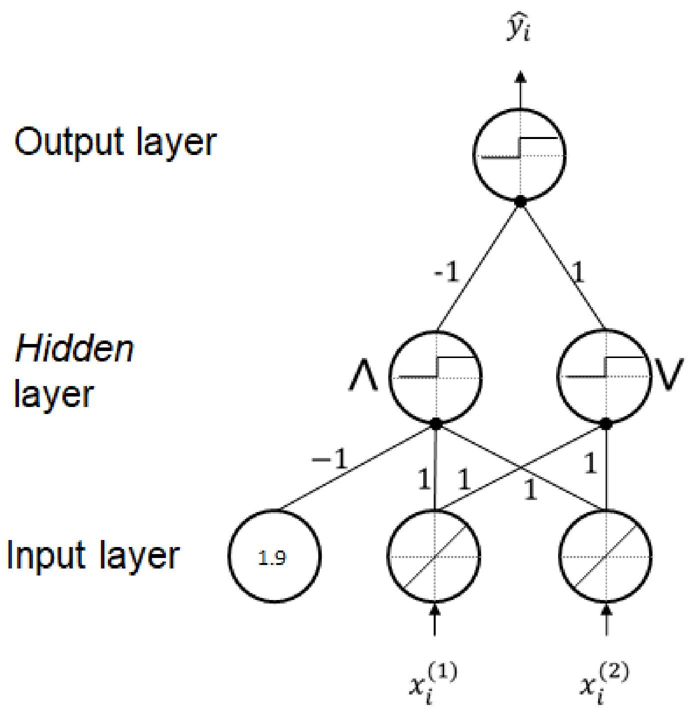
An MLP Solving the XOR Problem

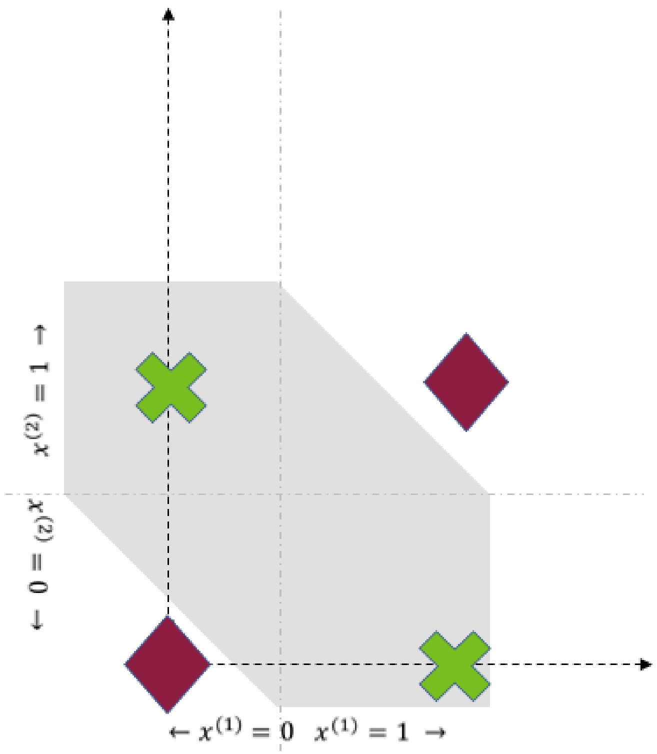




The Question of Learning

- **How can the network learn proper parameters from the given samples?**
  - Can the Perceptron algorithm be used?





Difficulty in Learning for MLP

Perceptron Learning Algorithm

- The weight update of the neuron is proportional to the “error” computed as $y_i - \hat{y}_i$.
  - This requires us to know the target output y_i.

Multi-layer Perceptron

- Except for the neurons on the output layer, other neurons (on the *hidden* layers) do not really have a target output given.

Back-propagation (BP) Learning for MLP

- **The key: Properly distribute error computed from output layer back to earlier layers to allow their weights to be updated in a way that reduce the error**
  - The basic philosophy of the BP algorithm

- **Differentiable activation functions**
  - We can use – e.g.,
    - the logistic neurons
    - neurons with sigmoid activation
    - or its variants

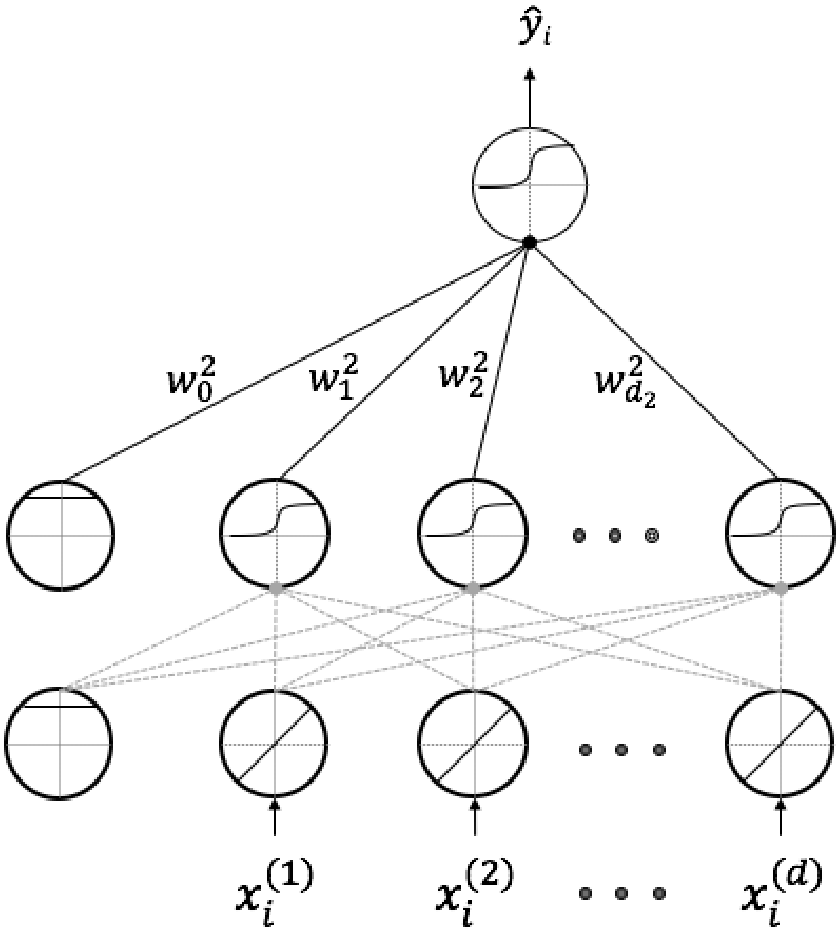
A Multi-Layer Neural Network

Using Logistic Neurons

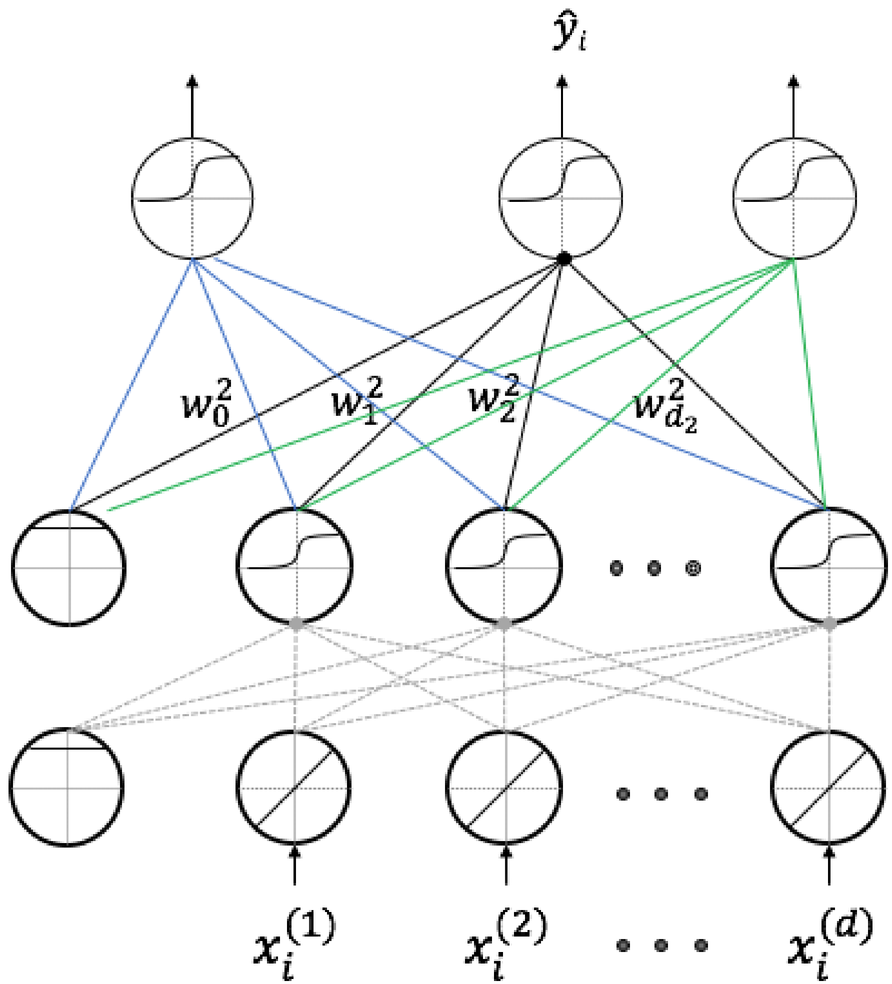
Output layer

Hidden layer

Input layer

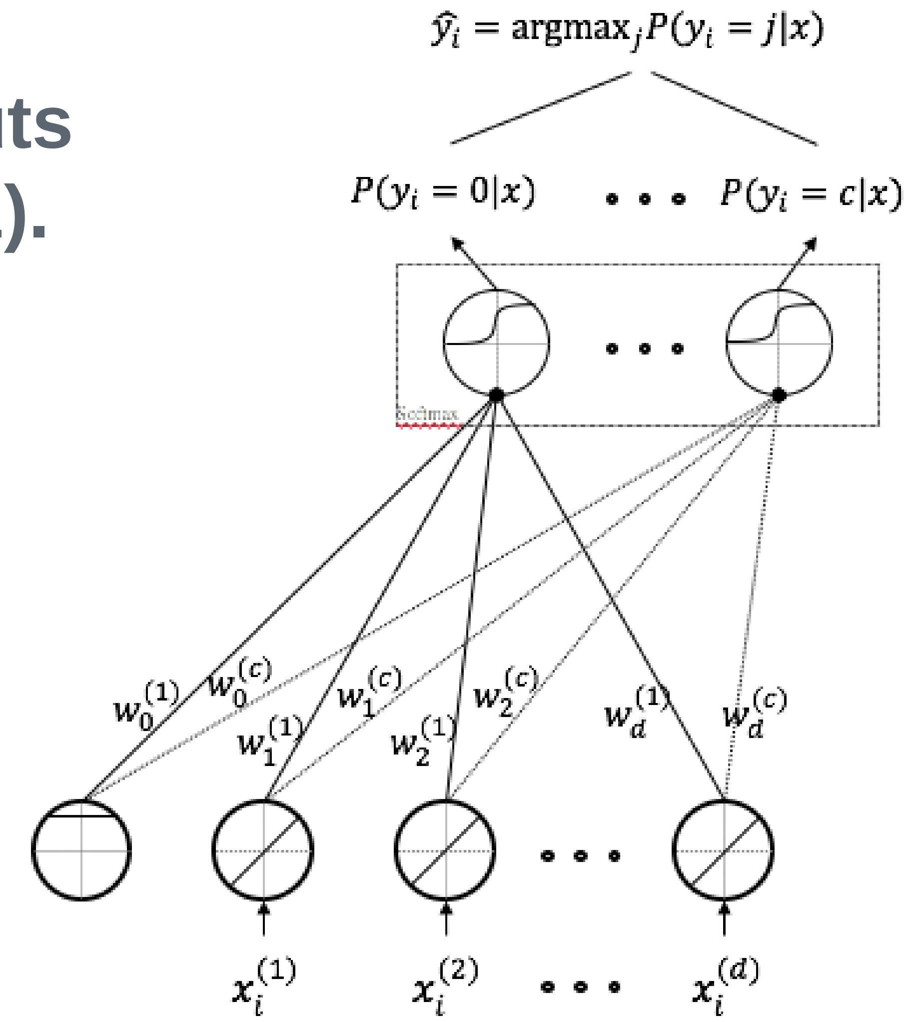


Handling Multiple (>2) Classes



Softmax for Handling Multiple Classes

- **Using *softmax* to normalize the outputs (so they add up to 1).**



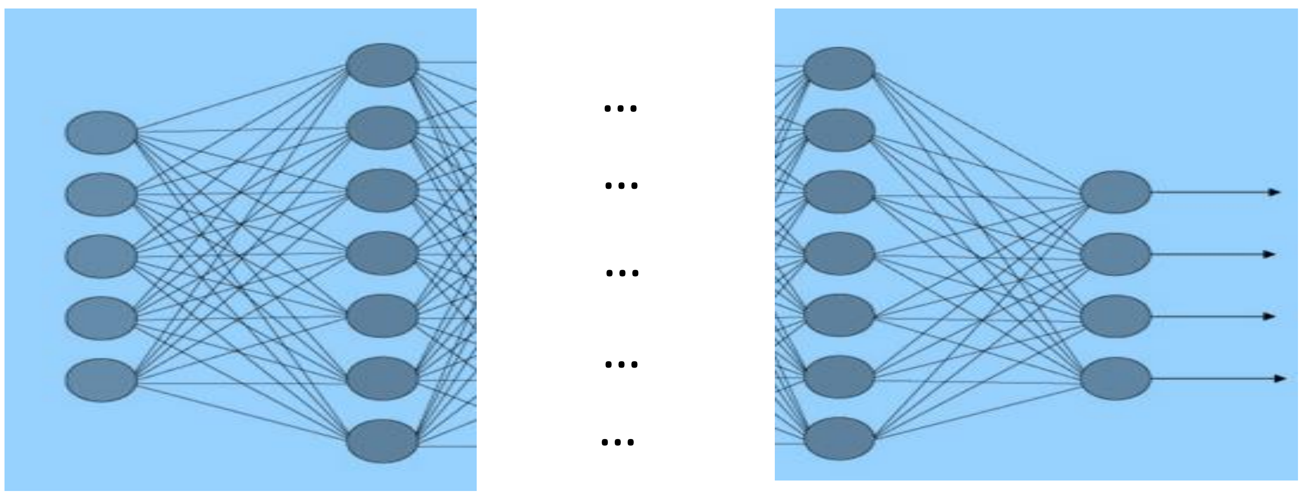
How to Compute “errors” in this Case?

Consider the cross-entropy as a loss function:

$$l(\boldsymbol{W}) = \sum_{i=1}^{n} \sum_{j=1}^{c} \mathbb{I}_j(y_i) \log P(y_i = j | x_i) \ ,$$

$$\mathbb{I}_j(y_i) = \begin{cases} 1, & if\ y_i = j \\ 0, & otherwise \end{cases}$$

Neural Networks and Deep Learning



Introduction to Deep Learning

Key Techniques Enabling Deep Learning

Ira A. Fulton Schools of Engineering
Arizona State University

Objective

Objective

Explain how, in principle, learning is achieved in a deep network

Objective

Explain key techniques that enable efficient learning in deep networks

Overview

- Back-propagation algorithm
- Design of activation functions
- Regularization for improving performance

* Technological advancement in computing hardware is certainly another enabling factor but our discussion will focus on basic, algorithmic techniques.

Back Propagation (BP) Algorithm

- **Simple Perceptron algorithm illustrates a path to learning by iterative optimization**
  - Updating weights based on network errors under current weights, and optimal weights are obtained when errors become 0 (or small enough)
- **Gradient descent is a general approach to iterative optimization**
  - Define a loss function J
  - Iteratively update the weights W according to the gradient of J with respect to W.

$\mathbf{W} \leftarrow \mathbf{W} - \eta \triangledown J(\mathbf{W})$ W is the parameter of the network; J is the objective function.

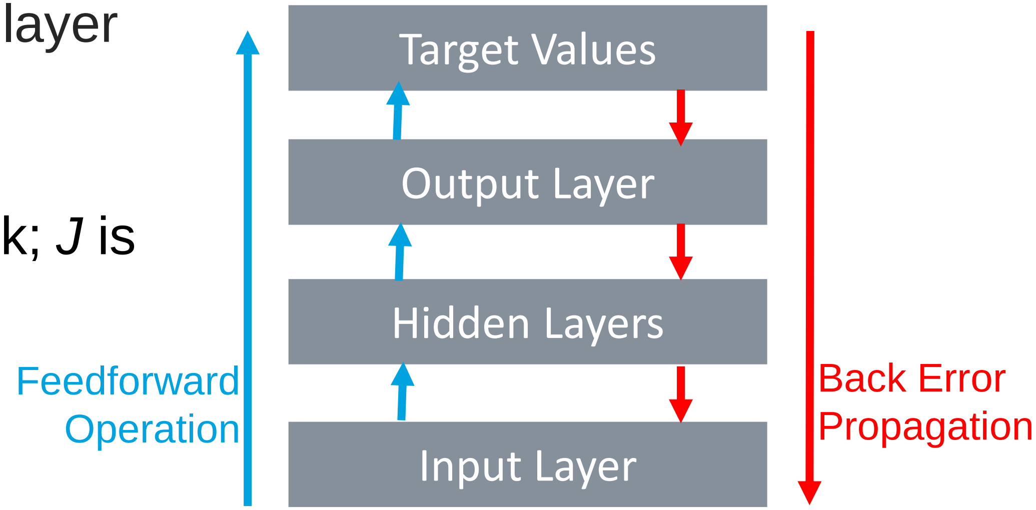
Back Propagation (BP) Algorithm (cont'd)

Generalizes/Implements the idea for multi-layer networks

- Gradient descent for updating weights in optimizing a loss function
- Propagating gradients back through layers
  - hidden layer weights are linked to loss gradient at output layer

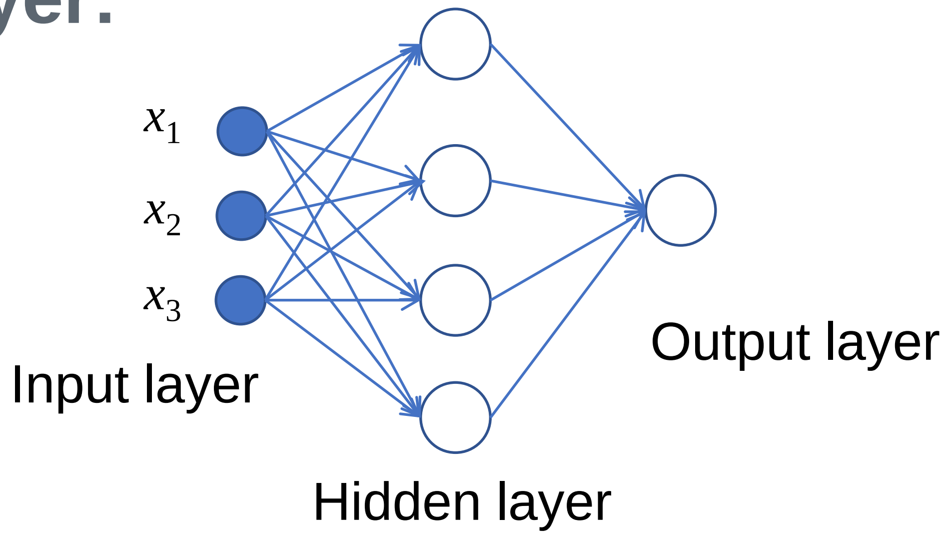
$$\mathbf{W} \leftarrow \mathbf{W} - \eta \bigtriangledown J(\mathbf{W})$$

W is the parameter of the network; *J* is the objective function.



Illustrating the BP Algorithm 1/6

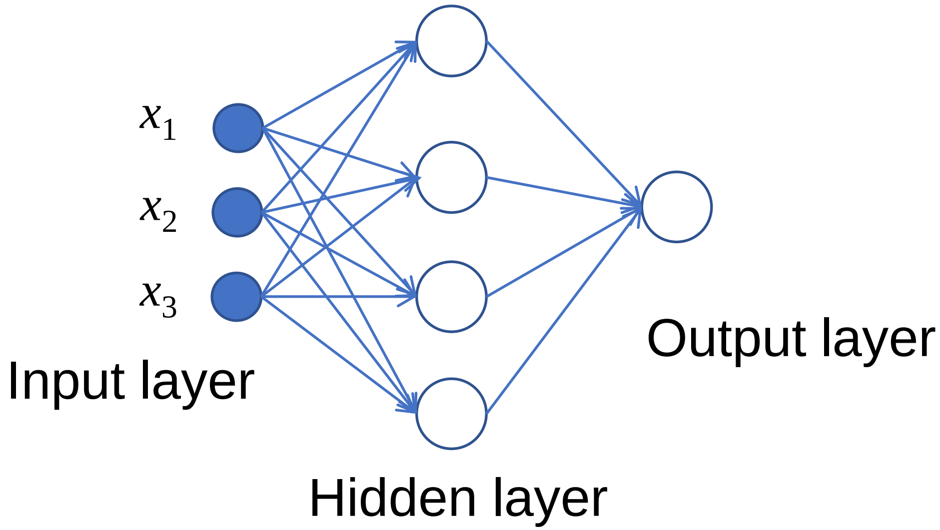
- **Let's consider a simple neural network with a single hidden layer. (We will only outline the key steps.)**
  - Let's write the net input and activation for a hidden node:
- **Let's write the net input and activation for the hidden layer:**



Illustrating the BP Algorithm 2/6

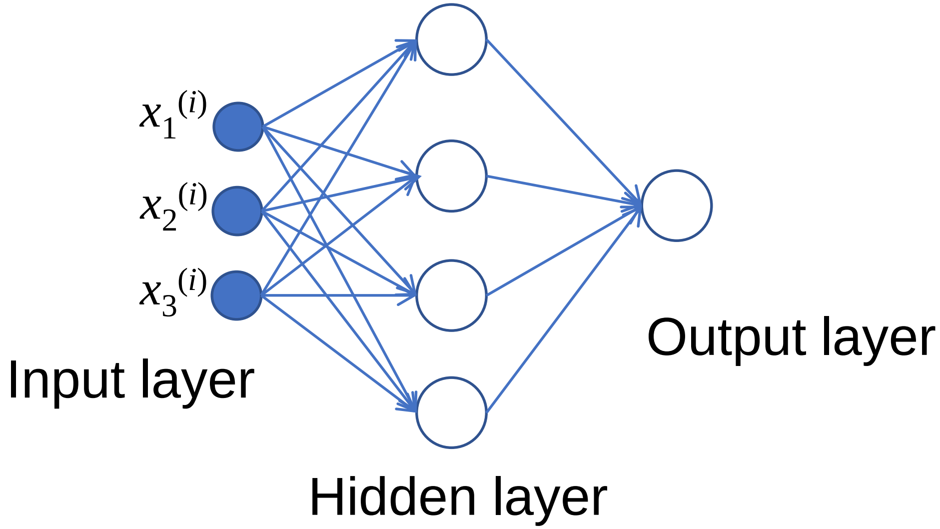
- Using matrix/vector notations, for the hidden layer:

- Similarly, for the output layer → Homework.



Illustrating the BP Algorithm 3/6

- Now consider m samples as input.
- Output layer is similarly done.



Illustrating the BP Algorithm 4/6

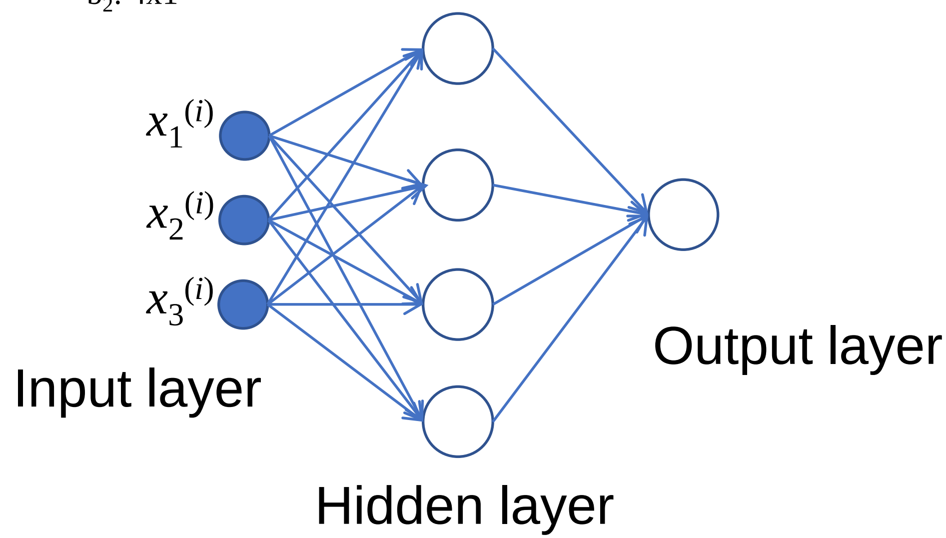
- Overall we have this flow of *feedforward* processing (note the notation change for simplicity: subscripts are for layers):

$$\xrightarrow[3\mathrm{xm}]{A_0=X} \boxed{Z_1=W_1A_0+b_1} \xrightarrow[4\mathrm{xm}]{Z_1} \boxed{A_1=f(Z_1)} \xrightarrow[4\mathrm{xm}]{A_1} \boxed{Z_2=W_2A_1+b_2} \xrightarrow[1\mathrm{xm}]{Z_2} \boxed{A_2=f(Z_2)} \xrightarrow[1\mathrm{xm}]{A_2} \boxed{L(A_2,Y)}$$

W_1: 4x3
b_1: 4x1

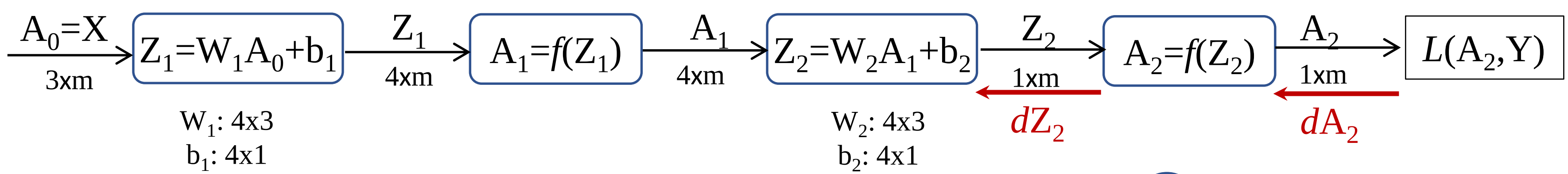
W_2: 4x3
b_2: 4x1

- **Consider** $\mathbf{dW_2} \triangleq \dfrac{\partial L}{\partial W_2}$

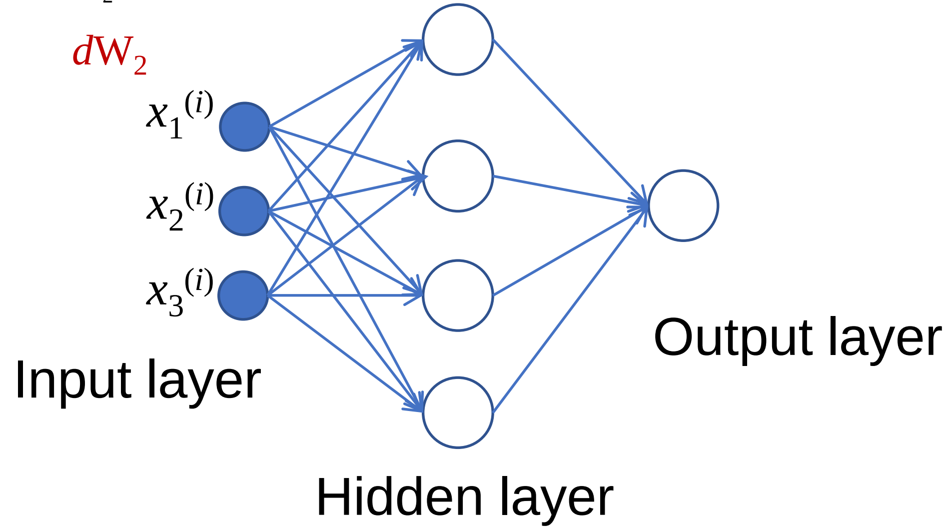


Illustrating the BP Algorithm 5/6

Back-propagation

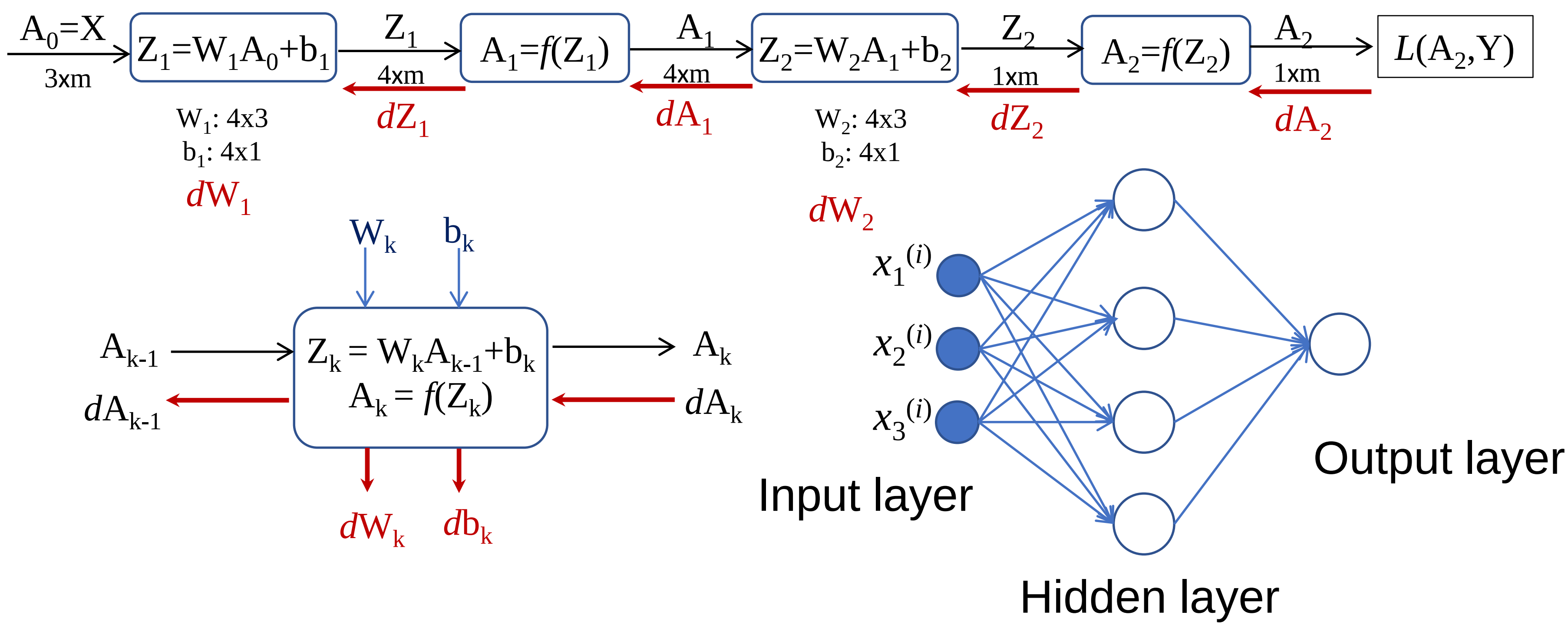


Consider $dW_1 \triangleq \frac{\partial L}{\partial W_1}$



Illustrating the BP Algorithm 6/6

A modular view of the layers



BP Algorithm Recap

- **The feedforward process: ultimately produce $A^{[K]}$ that leads to the prediction for Y.**
- **The backpropagation process:**
  - First compute the loss
  - Then compute the gradients via back-propagation through layers
  - Key: use the chain rule of differentiation
- **Essential to deep networks**
- **Suffers from several practical limitations**
  - gradient exploding
  - gradient vanishing
  - etc.
- **Many techniques were instrumental to enabling learning with BP algorithm for deep neural networks**

Activation Functions: Importance

- Provides non-linearity
- Functional unit of input-output mapping
- Its form impacts on gradients in BP algorithm

Activation Functions: Choices

Older Types

- Thresholding
- Logistic function
- tanh

Newer Types

- Rectifier $f(x) = \max(0, x)$ and its variants
- Rectified Linear Unit (ReLU)

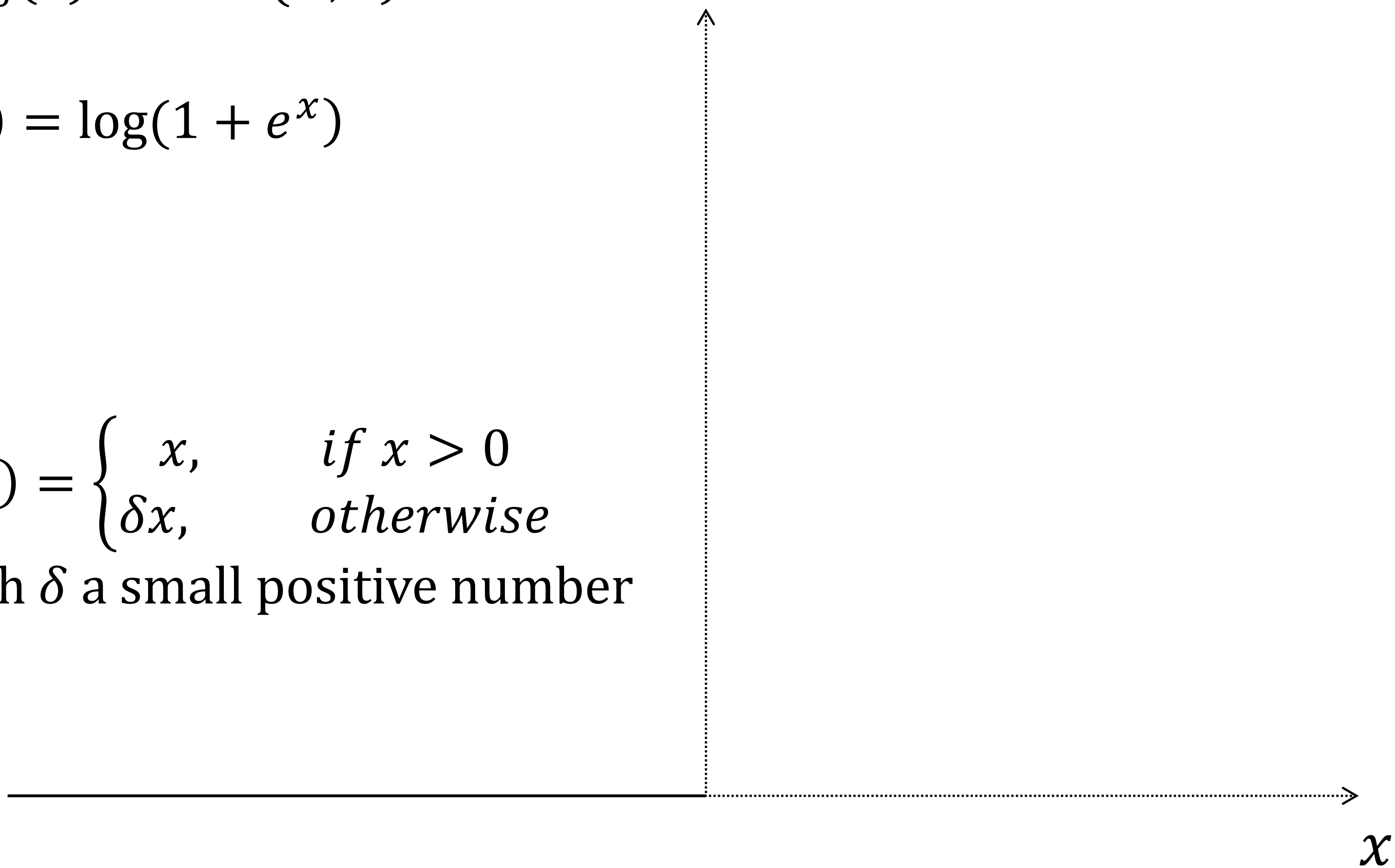
ReLU and Some Variants

$$a_{\mathrm{ReLU}}(x) = \max(0, x)$$

$$a_{\mathrm{s}}(x) = \log(1 + e^{x})$$

$$a_{\mathrm{L}}(x) = \begin{cases} x, & if\ x > 0 \\ \delta x, & otherwise \end{cases}$$

with δ a small positive number



The Importance of Regularization

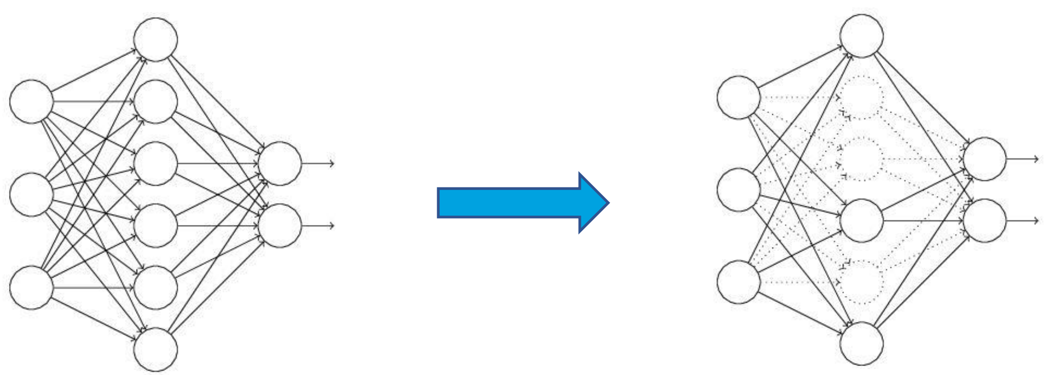
- **The parameter space is huge, if there is no constraint in search for a solution, the algorithm may converge to poor solutions.**
- **Overfitting is a typical problem**
  - Converging to local minimum good only for the training data

Some Ideas for Regularization

- **Favoring a network with small weights**
  - achieved by adding a term of L2-norm of the weights to original loss function
- **Preventing neurons from "co-adaptation" ➔ Drop-out**
- **Making the network less sensitive to initialization/learning rate etc.**
  - Batch normalization
- **Such regularization techniques have been found to be not only helpful but sometimes critical to learning in deep networks**

Drop-out

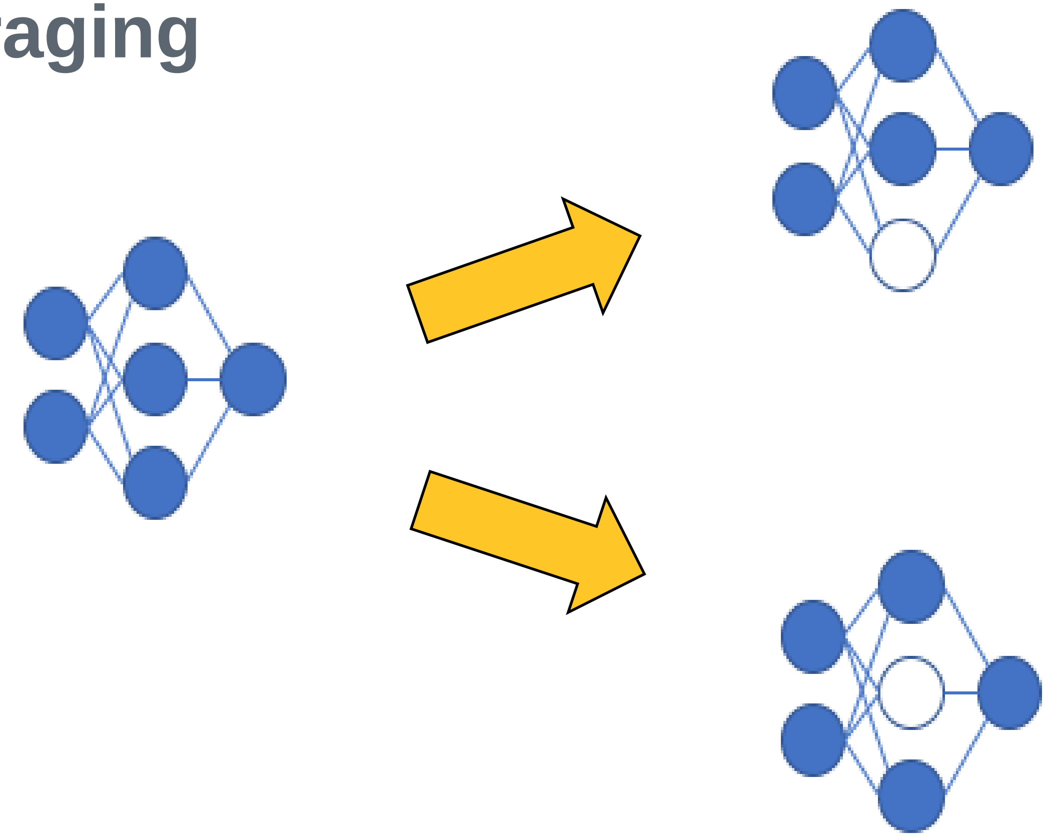
- Obtain (b) by randomly deactivate some hidden nodes in (a)
- For input x, calculate output y by using the activated nodes ONLY
- Use BP to update weights (which connect to the activated nodes) of network
- Activate all nodes
- Go back to first step



(a) (b)

Why Drop-out?

- Reducing co-adaptation of neuron
- Model averaging



Batch Normalization (BN)

- **Inputs to network layers are of varying distributions, the so-called internal covariate shift [Ioffe and Szegedy, 2015]**
  - Careful parameter initialization and low learning rate are required
- **BN was developed to solve this problem by normalizing layer inputs of a batch**

The Simple Math of BN

For a mini-batch with size = m, first calculate

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

Up to this point, $\widehat{x}$ has mean = 0 and standard deviation = 1

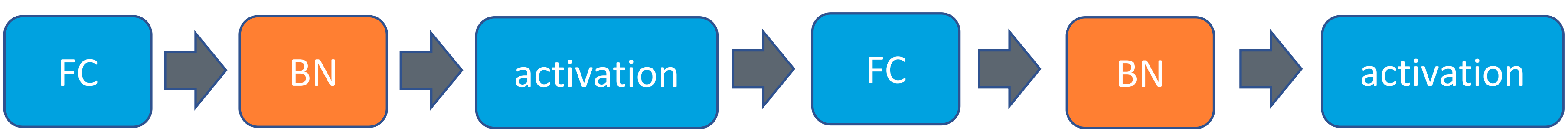
How is BN Used in Learning?

- Define two parameters and so that the output of the BN layer can be calculated as:

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$$

- Parameters β and γ can be learned by minimizing the lost function via gradient descent

- Usually used right before the activation functions



Other Regularization Techniques

- **Weight sharing**
- **Training data conditioning**
- **Sparsity constraints**
- **Ensemble methods (committee of networks)**

➔ Some of these will be discussed in later examples of networks.

Introduction to Deep Learning

Some Basic Deep Architecture

Ira A. Fulton Schools of Engineering
Arizona State University

Objective

Objective

Appraise the detailed architecture of a basic convolutional neural network

Objective

Explain the basic concepts and corresponding architecture for auto-encoders and recurrent neural networks

Overview

- **Convolutional Neural Network (CNN)**
  - will be given the most attention, for its wide range of application
- **Auto-encoder**
- **Recurrent Neural Networks (RNN)**

Convolutional Neural Network (CNN)

- Most useful for input data defined on grid-like structures, like images or audio
- Built upon concept of “convolution” for signal/image filtering
- Invokes other concepts like pooling, weight-sharing, and (visual) receptive field, etc.

Image Filtering via Convolution 1/5

Kernel coefficients

$$\frac{1}{12}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 4 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Kernel or mask

image

255	128	128	240	1	128	24	255
255	128	245	240	1	128	24	255
245	128	245	240	1	128	128	128
24	24	24	255	255	0	0	0
0	0	0	0	0	0	0	0
245	245	128	128	128	128	245	245
245	245	245	240	240	240	255	240
255	240	128	240	240	128	240	255

Image Filtering via Convolution 2/5

New pixel value =
(1*255+
1*128+
1*128+
1*255+
4*128+
1*245+
1*245+
1*128+
1*245)/12=
2141/12
=178

255 1	128 1	128 1	240	1	128	24	255
255 1	128 4	245 1	240	1	128	24	255
245 1	128 1	245 1	240	1	128	128	128
24	24	24	255	255	0	0	0
0	0	0	0	0	0	0	0
245	245	128	128	128	128	245	245
245	245	245	240	240	240	255	240
255	240	128	240	240	128	240	255

Image Filtering via Convolution 3/5

255	128	128	240	1	128	24	255
255	**178**	245	240	1	128	24	255
245	128	245	240	1	128	128	128
24	24	24	255	255	0	0	0
0	0	0	0	0	0	0	0
245	245	128	128	128	128	245	245
245	245	245	240	240	240	255	240
255	240	128	240	240	128	240	255

Image Filtering via Convolution 4/5

New pixel value = (1*128+ 1*128+ 1*240+ 1*178+ 4*245+ 1*240+ 1*128+ 1*245+ 1*240)/12= 2507/12 =209

255	128 1	128 1	240 1	1	128	24	255
255	**178** 1	245 4	240 1	1	128	24	255
245	128 1	245 1	240 1	1	128	128	128
24	24	24	255	255	0	0	0
0	0	0	0	0	0	0	0
245	245	128	128	128	128	245	245
245	245	245	240	240	240	255	240
255	240	128	240	240	128	240	255

Image Filtering via Convolution 5/5

255	128	128	240	1	128	24	255
255	**178**	**209**	240	1	128	24	255
245	128	245	240	1	128	128	128
24	24	24	255	255	0	0	0
0	0	0	0	0	0	0	0
245	245	128	128	128	128	245	245
245	245	245	240	240	240	255	240
255	240	128	240	240	128	240	255

Image Filtering via Convolution: Kernels

- **By varying coefficients of the kernel, we can achieve different goals**
  - Smoothing, sharpening, detecting edges, etc.
- **Better yet: can we learn proper kernels? Part of CNN objective**
- **Examples of Kernels:**

1/12

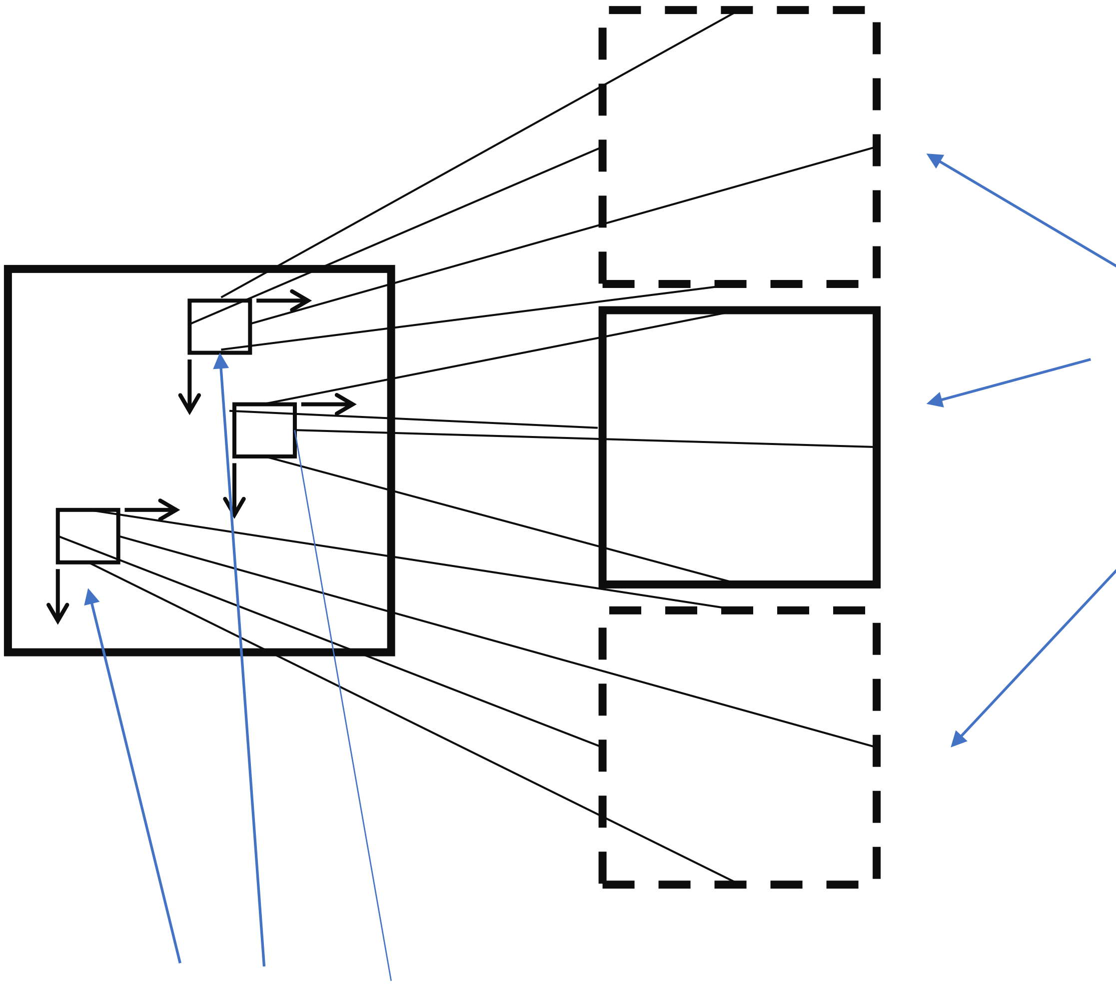
1	1	1
1	4	1
1	1	1

Smoothing/Noise-reduction

1	0	-1
1	0	-1
1	0	-1

(Vertical) Edge detection

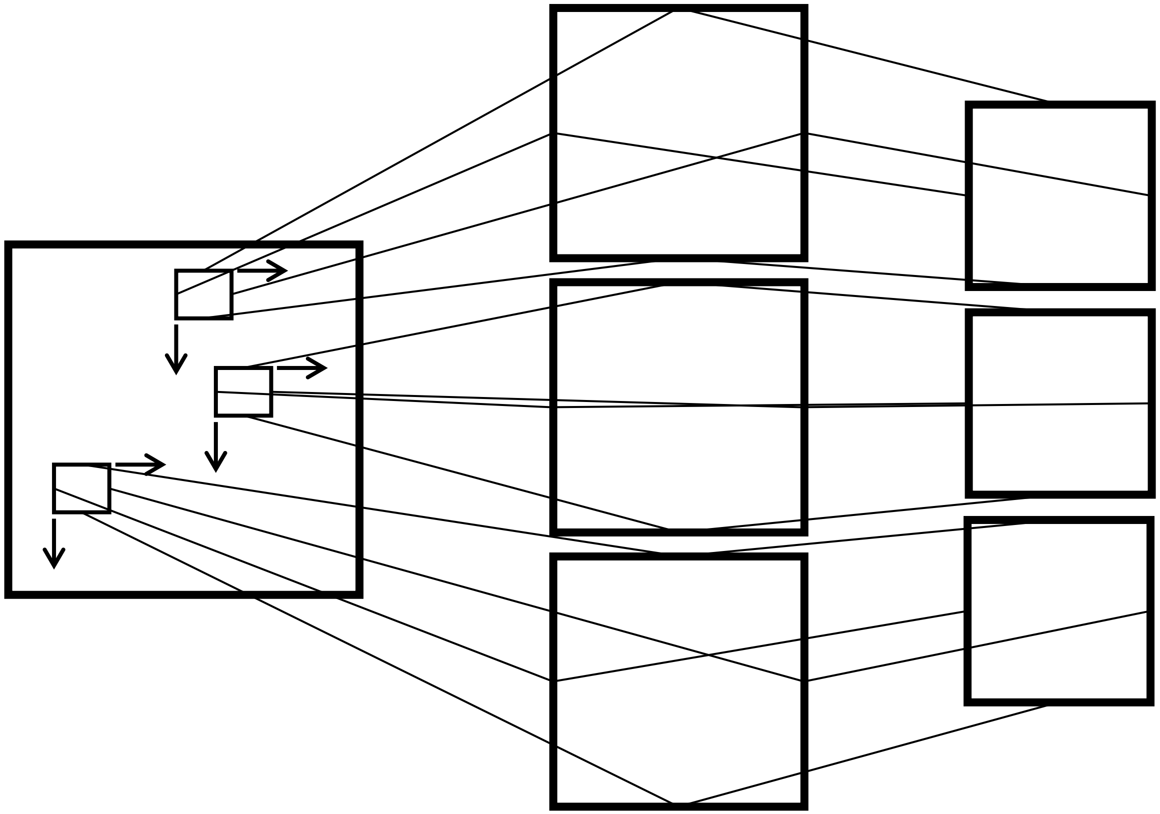
2D Convolutional Neuron



The number of kernels defines the number of *features maps* (*channels*).

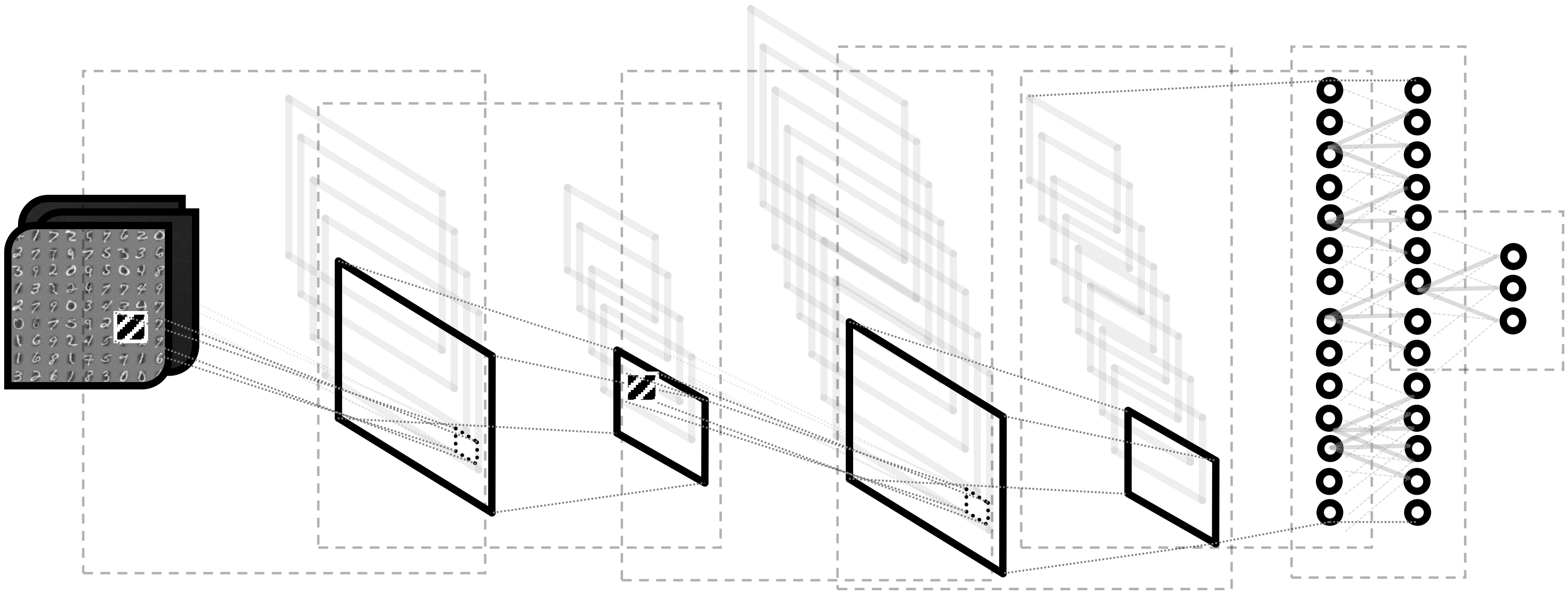
The sizes of the kernels define the *receptive fields*.

Convpool Layer



Convolution, pooling, and going through some activations

Illustrating A Simple CNN



Some convpool layers plus some fully-connected layers

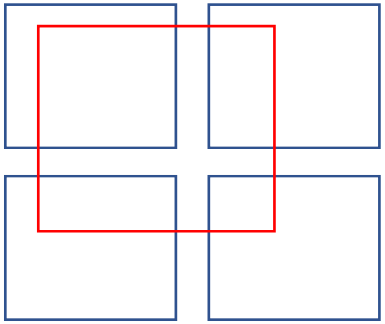
CNN Examples - Different Complexities

- LeNet
- AlexNet

LeNet

- Each pixel in layer 3 corresponds to 7/3 of a pixel in the input Second level
- Receptive field of layer 1 is 5X5

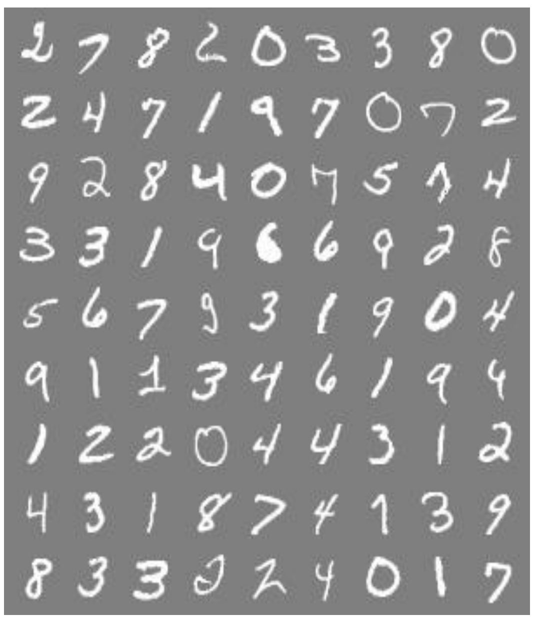
Layer Number	Input Shape	Receptive Field	Number of Feature Maps	Type of Neuron
1	28 X 28 X 1	5 X 5	20	Convolutional
2	24 X 24 X 20	2 X 2		Pooling
3	12 X 12 X 20	5 X 5	50	Convolutional
4	8 X 8 X 50	2 X 2		Pooling
5	800	1 X 1	500	Fully Connected
6	500		10	Softmax

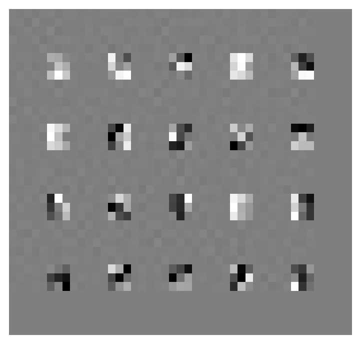


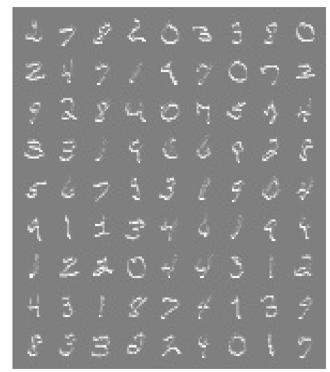
LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998): 2278-2324.

Case Study: LeNet

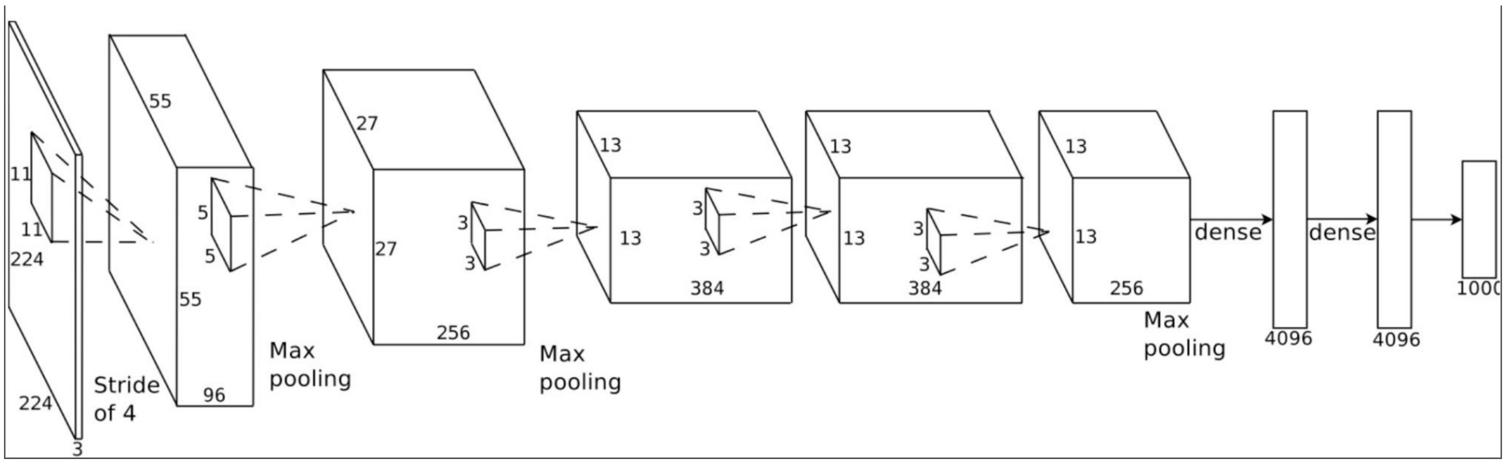
- This is after training the network for 75 epochs with a learning rate of 0.01
- Produces an accuracy of 99.38% on the MNIST dataset.

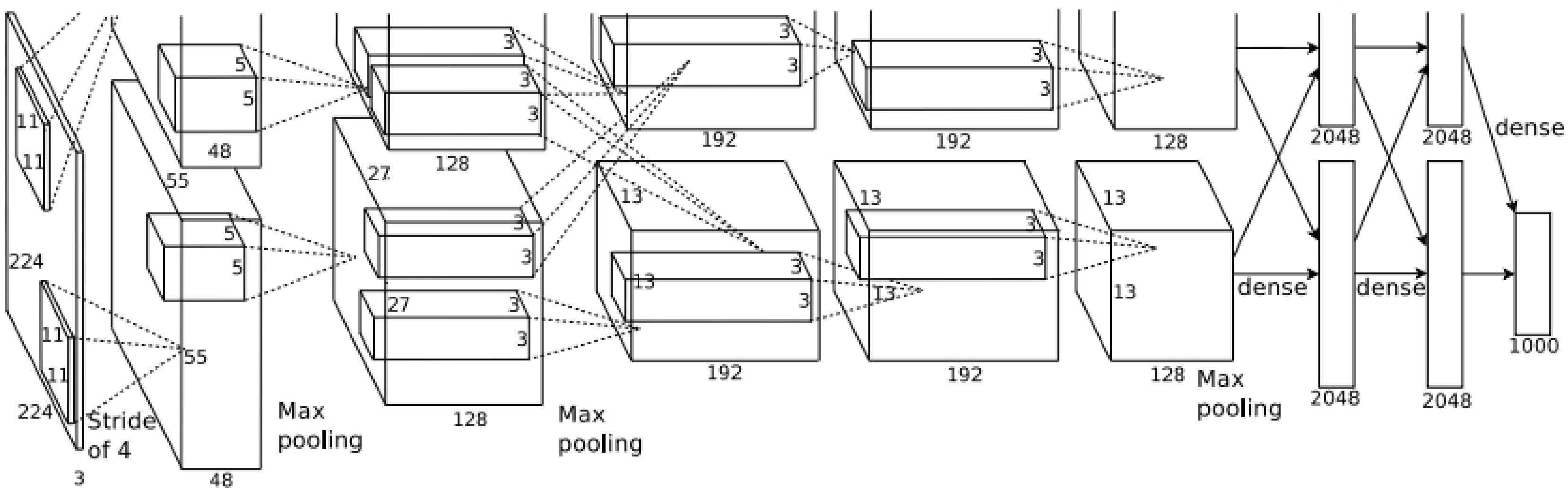






Case Study: AlexNet





Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

Case Study: AlexNet 1/3

Layer Number	Input Shape	Receptive Field	Number of Kernels	Type of Neuron
1	224 X 224 X 3	11 X 11, stride 4	96	Convolutional
2		3 X 3, stride 2		Pooling
3	55 X 55 X 96	5 X 5	256	Convolutional
4		3 X 3, stride 2		Pooling
5	13 X 13 X 256	3 X 3, padded	384	Convolutional
6	13 X 13 X 384	3 X 3, padded	384	Convolutional
7	13 X 13 X 384	3 X 3	256	Convolutional
8	43264	1 X 1	4096	Fully Connected
9	4096	1 X 1	4096	Fully Connected
10	4096		1000	Softmax

Receptive field of the layer 7 is

~ 52 pixels !! which is almost as big as an object part (about one – fourth of the input image)

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

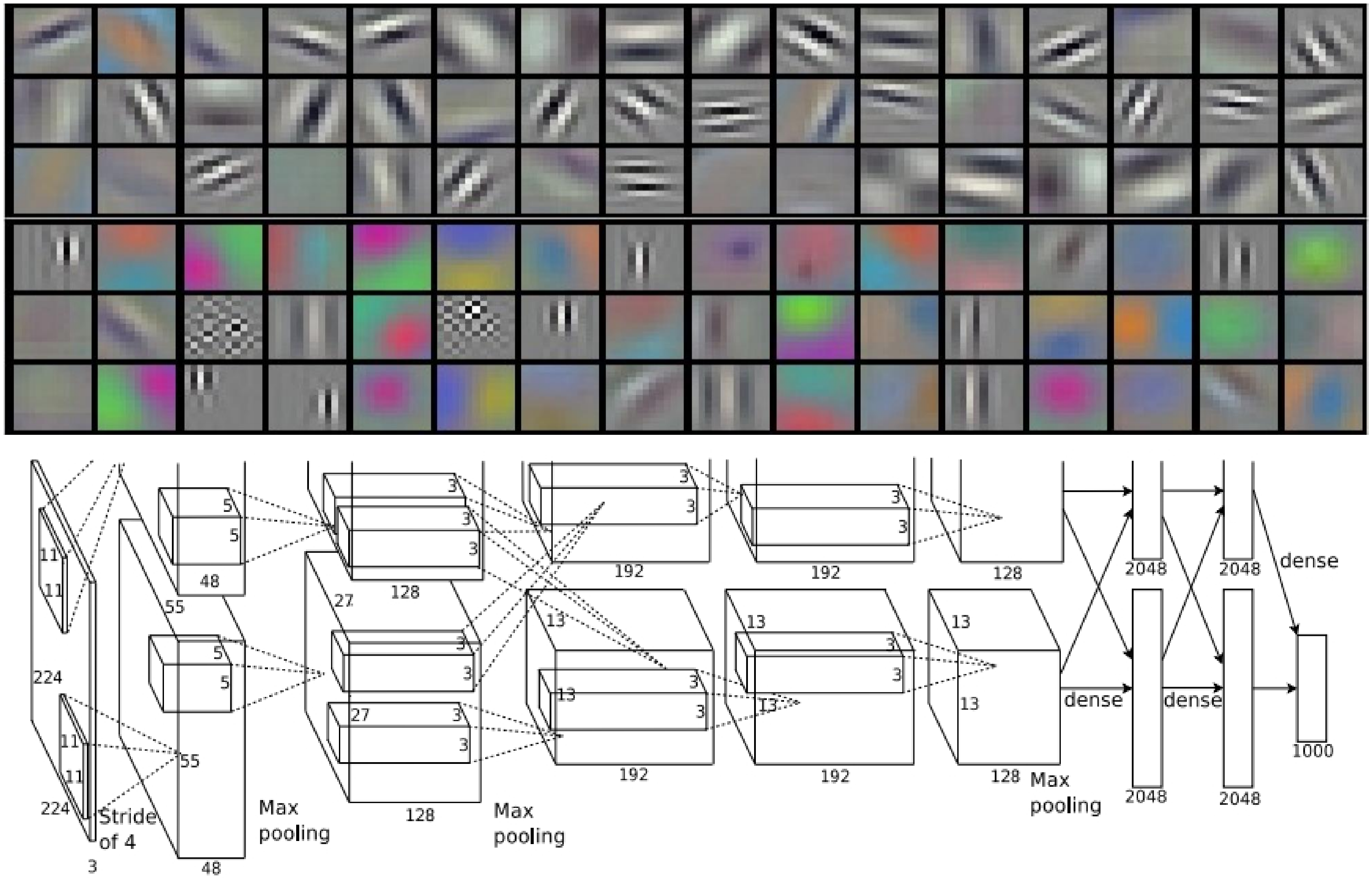
Case Study: AlexNet 2/3

- **Imagenet -15 million images in over 22,000 categories**
- **Imagenet categories are much more complicated than other datasets**
  - Often difficult even for humans to categorize perfectly
  - Average human-level performance is about 96% on this dataset
- **(ILSVRC), used about 1000 of these categories**
- **AlexNet was the earliest systems to break the 80% mark**
  - Non-neural conventional techniques were unable to achieve such performance

Case Study: AlexNet 3/3

- **AlexNet was huge at the time.**
  - The size could lead to instability during training or inability to learn, if without proper regularization
- **Some techniques were used to make it trainable**
  - AlexNet was the first prominent network to feature ReLU
  - Features multi-GPU training (originally trained the networks on two Nvidia GTX 580 GPUs with 3GB)

Case Study: AlexNet Filters



Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

CNN Recap

- **The CNNs are similar to the basic MLP architecture illustrated earlier, but some key extensions include:**
  - The concept of weight-sharing through kernels
  - Weight-sharing enables learnable kernels, which in turn define feature maps
  - The idea of pooling

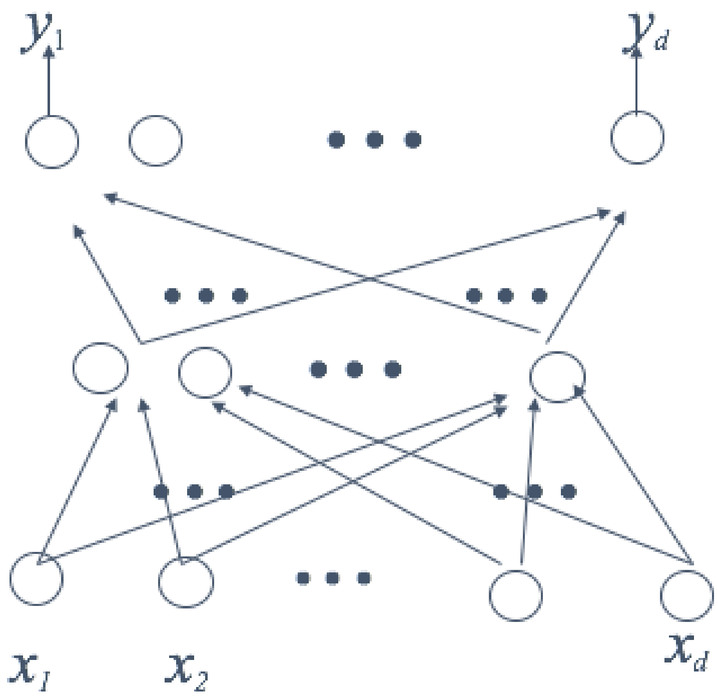
Auto-encoder 1/4

- **Networks seen thus far are all trained via supervised learning**

- **Sometimes we may need to train a network without supervision:**

  → Unsupervised learning

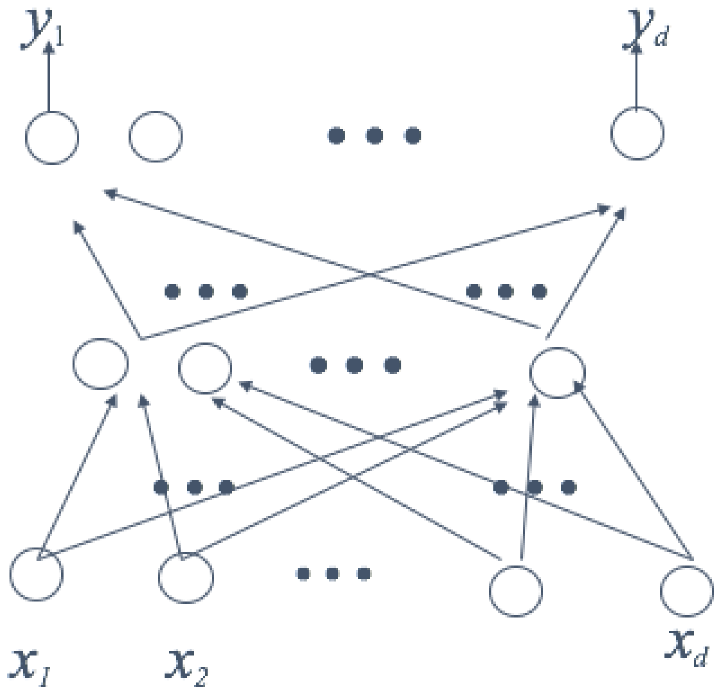
- **Auto-encoder is a such example**
  - Consider y_i being an approximation of x_i.



Auto-encoder 2/4

- Perfect auto-encoder would map x_i to x_i

- Learn good representations in the hidden layer



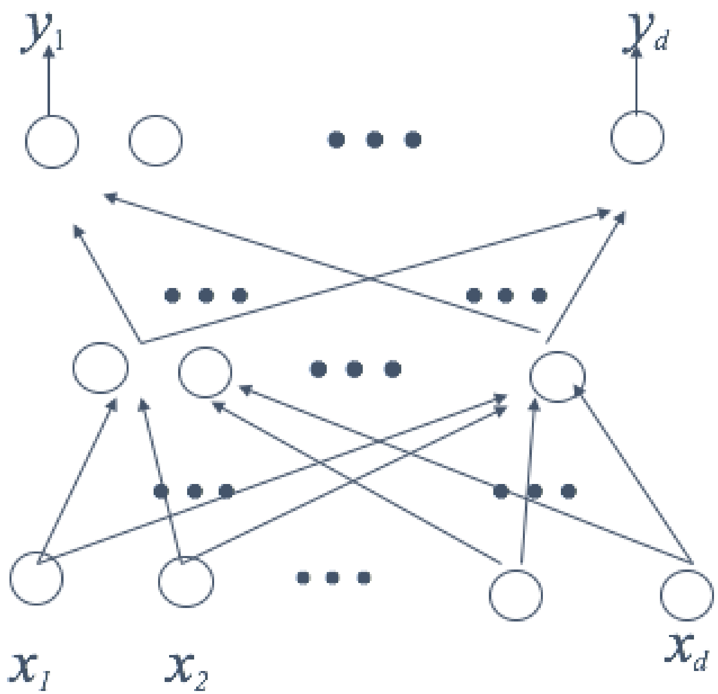
Auto-encoder 3/4

Consider two cases

- Much fewer hidden nodes than input nodes
- Many hidden nodes or more hidden nodes than input nodes

Case 1: Encoder for compressing input and compressed data should still be able to reconstruct the input

- Similar to, e.g., PCA



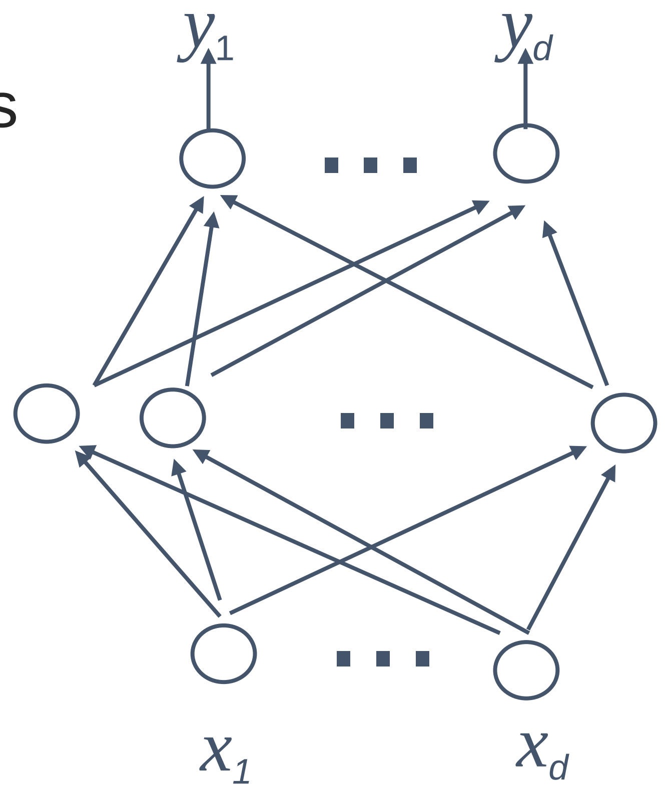
Auto-encoder 4/4

Consider two cases

- Fewer hidden nodes than input nodes
- More hidden nodes than input nodes

Case 2: Allow more hidden nodes than input

- Allow more freedom for the input-to-hidden layer mapping in exploring structure of the input
- Additional “regularization” will be needed in order to find meaningful results

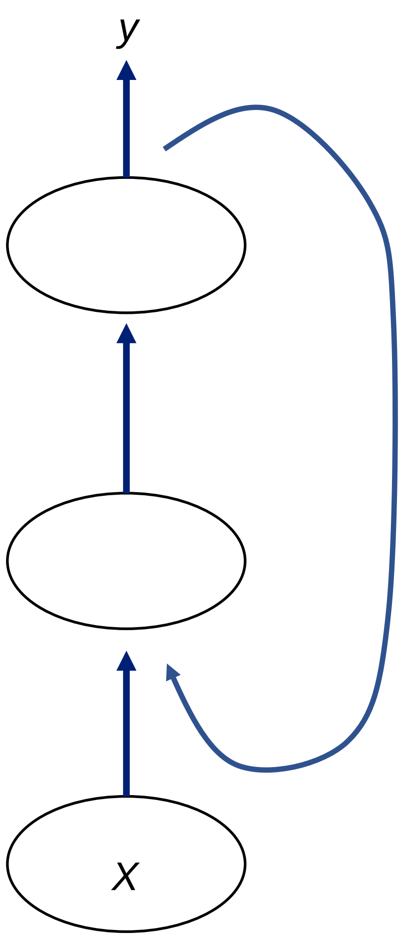


Recurrent Neural Networks (RNNs) 1/4

- ***Feedforward networks*: Neurons are interconnected without any cycle in the connection**
- ***Recurrent neural networks*: Allow directed cycles in connections between neurons**
  - Notion of “state” or temporal dynamics
  - Necessity of internal memory
- **One clear benefit: Such networks could naturally model variable-length sequential data**

Recurrent Neural Networks (RNNs) 2/4

- **A basic, illustrative architecture for RNN (showing only one node each layer)**
  - **QUESTION:** What is this network equivalent to, if we "unfold" the cycles for a given sequence of data?



Recurrent Neural Networks (RNNs) 3/4

- Training with BP algorithm may suffer from so-called *vanishing gradient problem*
- Some RNN variants have sophisticated "recurrence" structures, invented in part to address such difficulties faced by basic RNN models

Recurrent Neural Networks (RNNs) 4/4

- **Examples:**
- **The "Long short-term memory" (LSTM) model**
  - used to produce state-of-the-art results in speech and language applications
- **The Gated Recurrent Unit model, illustrated here:**