CS145: INTRODUCTION TO DATA MINING

6: Vector Data: Neural Network

Instructor: Yizhou Sun

yzsun@cs.ucla.edu

October 21, 2018

Methods to Learn: Last Lecture

	Vector Data	Set Data	Sequence Data	Text Data
Classification	**Logistic Regression; Decision Tree**; KNN **SVM**; NN			Naïve Bayes for Text
Clustering	K-means; hierarchical clustering; DBSCAN; Mixture Models			PLSA
Prediction	**Linear Regression** GLM*			
Frequent Pattern Mining		Apriori; FP growth	GSP; PrefixSpan	
Similarity Search			DTW	

Methods to Learn

	Vector Data	Set Data	Sequence Data	Text Data
Classification	**Logistic Regression; Decision Tree**; KNN **SVM**; **NN**			Naïve Bayes for Text
Clustering	K-means; hierarchical clustering; DBSCAN; Mixture Models			PLSA
Prediction	**Linear Regression** GLM*			
Frequent Pattern Mining		Apriori; FP growth	GSP; PrefixSpan	
Similarity Search			DTW	

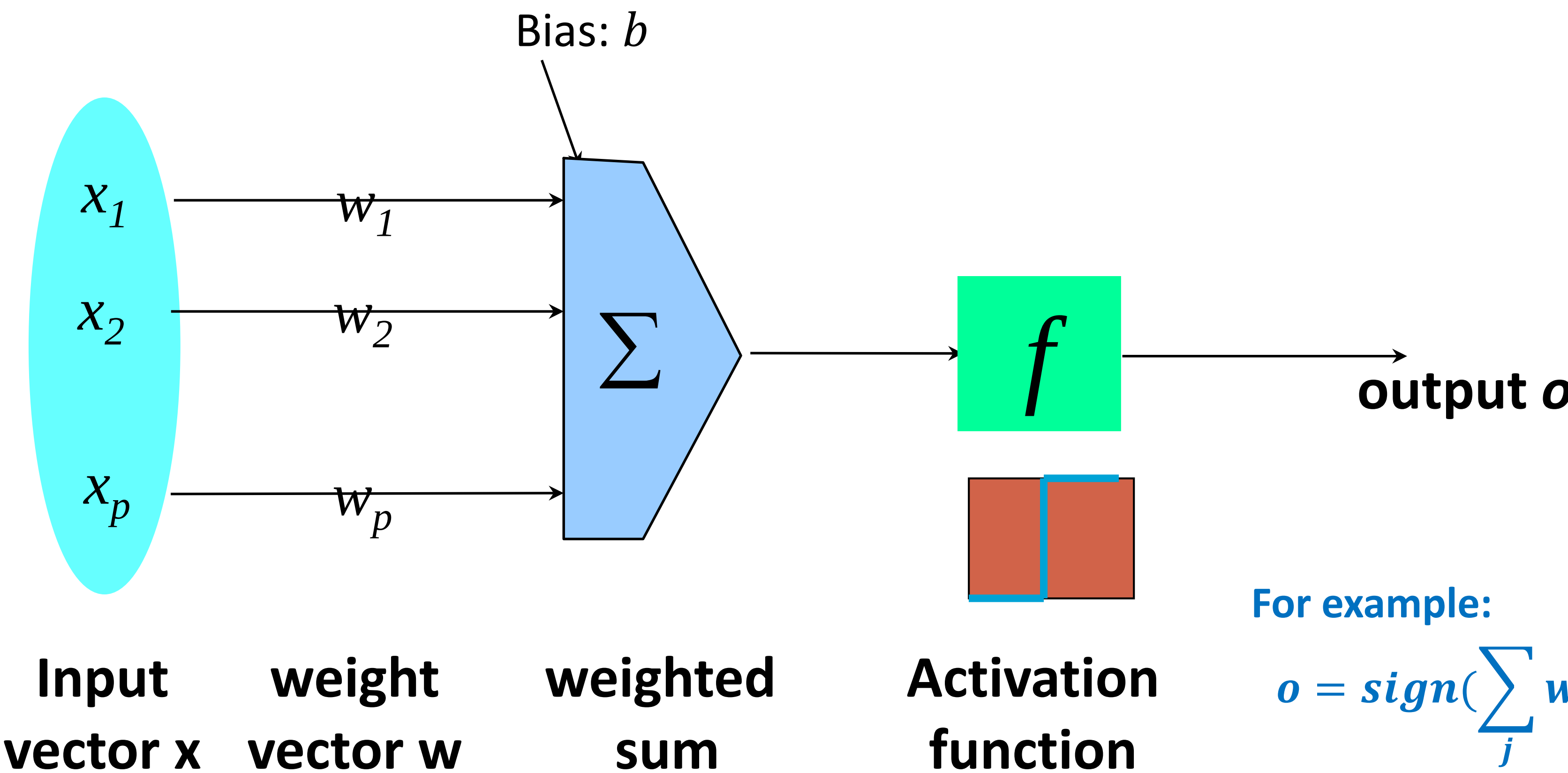
Neural Network

- Introduction
- Multi-Layer Feed-Forward Neural Network
- Summary

Artificial Neural Networks

- Consider humans:
  - Neuron switching time ~.001 second
  - Number of neurons ~10^{10}
  - Connections per neuron ~10^{4-5}
  - Scene recognition time ~.1 second
  - 100 inference steps doesn't seem like enough -> parallel computation
- Artificial neural networks
  - Many neuron-like threshold switching units
  - Many weighted interconnections among units
  - Highly parallel, distributed process
  - Emphasis on tuning weights automatically

Single Unit: Perceptron



For example:

$$o = sign(\sum_j w_j x_j + b)$$

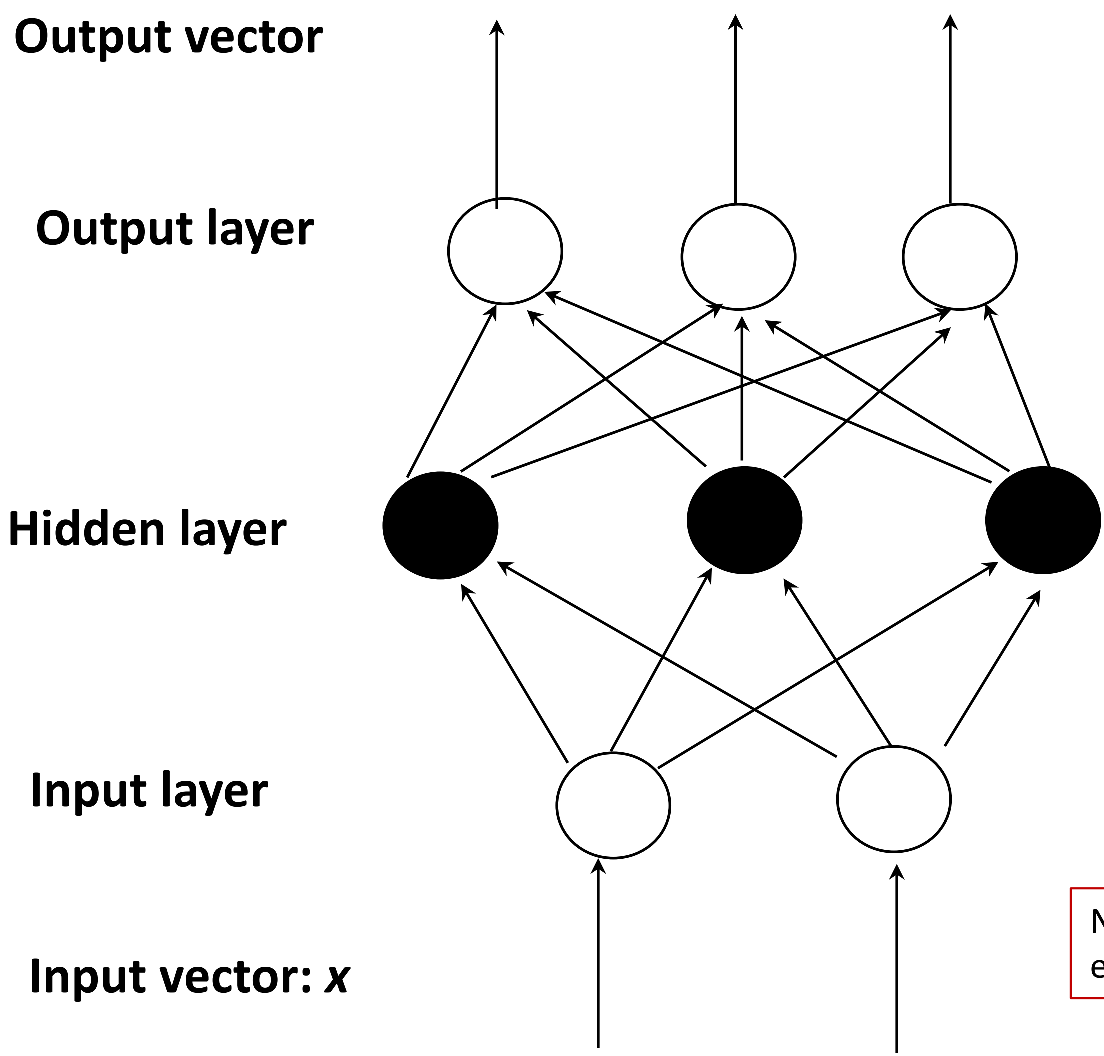
- An *n*-dimensional input vector **x** is mapped into variable y by means of the scalar product and a nonlinear function mapping

Neural Network

- Introduction
- Multi-Layer Feed-Forward Neural Network
- Summary

A Multi-Layer Feed-Forward Neural Network

A **two-layer** network



$$\boldsymbol{y} = g(W^{(2)}\boldsymbol{h} + b^{(2)})$$

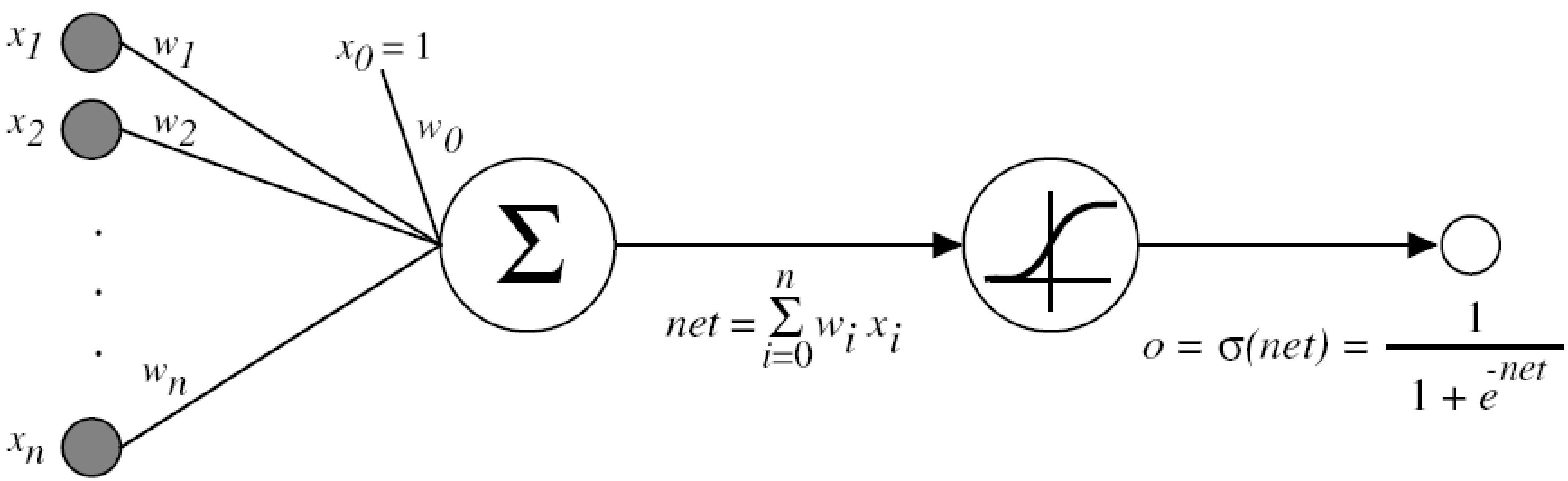
$$\boldsymbol{h} = f(W^{(1)}\boldsymbol{x} + b^{(1)})$$

Bias term

Weight matrix

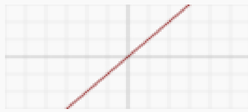
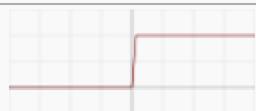
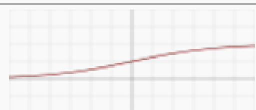
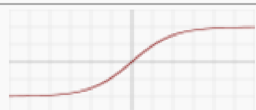
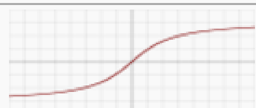
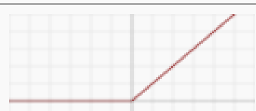
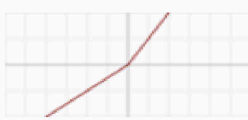
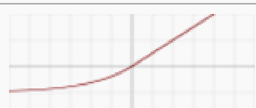
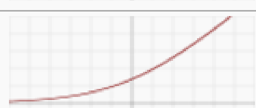
Nonlinear transformation,
e.g. sigmoid transformation

Sigmoid Unit



- $\sigma(x) = \frac{1}{1+e^{-x}}$ is a sigmoid function
  - Property: $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$
  - Will be used in learning

Activation Functions

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

How A Multi-Layer Neural Network Works

- The **inputs** to the network correspond to the attributes measured for each training tuple
- Inputs are fed simultaneously into the units making up the **input layer**
- They are then weighted and fed simultaneously to a **hidden layer**
  - The number of hidden layers is arbitrary
- The weighted outputs of the last hidden layer are input to units making up the **output layer**, which emits the network's prediction
- The network is **feed-forward**: None of the weights cycles back to an input unit or to an output unit of a previous layer
- From a math point of view, networks perform **nonlinear regression**: **Given enough hidden units and enough training samples, they can closely approximate any continuous function**

Defining a Network Topology

- Decide the **network topology:** Specify # of units in the *input layer*, # of *hidden layers* (if > 1), # of units in *each hidden layer*, and # of units in the *output layer*
- Normalize the **input** values for each attribute measured in the training tuples
- **Output**, if for classification and more than two classes, one output unit per class is used
- Once a network has been trained and its accuracy is **unacceptable**, repeat the training process with a different network topology or a different set of initial weights

Learning by Backpropagation

- Backpropagation: A **neural network** learning algorithm
- Started by psychologists and neurobiologists to develop and test computational analogues of neurons
- During the learning phase, the **network learns by adjusting the weights** so as to be able to predict the correct class label of the input tuples
- Also referred to as **connectionist learning** due to the connections between units

Backpropagation

- Iteratively process a set of training tuples & compare the network's prediction with the actual known target value
- For each training tuple, the weights are modified to **minimize the loss function** between the network's prediction and the actual target value, say **mean squared error**
  - Stochastic gradient descent + chain rule
- Modifications are made in the "**backwards**" direction: from the output layer, through each hidden layer down to the first hidden layer, hence "**backpropagation**"

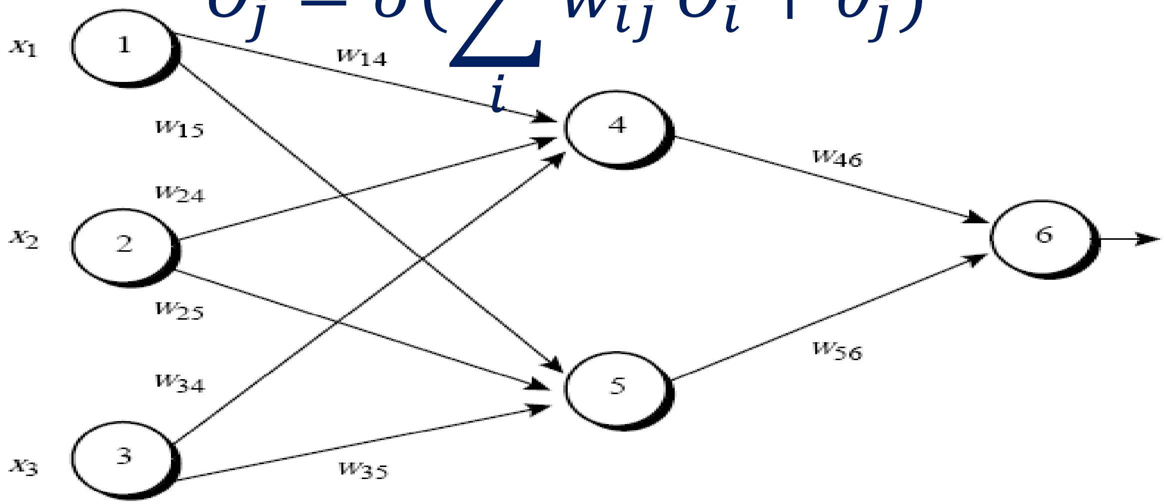
Example of Loss Functions

- Hinge loss
- Logistic loss
- Cross-entropy loss
- square error loss
- absolute error loss

A Special Case

- Activation function: Sigmoid

$$O_j = \sigma\left(\sum_i w_{ij}\, O_i + \theta_j\right)$$



- Loss function: square error loss

$$J = \frac{1}{2}\sum_j \left(T_j - O_j\right)^2, \; for\, j\, in\, output\, layer$$

$$T_j: true\ value\ of\ output\ unit\ j;$$

$$O_j: output\ value$$

Backpropagation Steps to Learn Weights

- Initialize weights to small random numbers, associated with biases
- Repeat until terminating condition meets
  - **For** each training example
    - **Propagate the inputs forward** (by applying activation function)
      - For a hidden or output layer unit j
        - Calculate net input: $I_j = \sum_i w_{ij} O_i + \theta_j$
        - Calculate output of unit j: $O_j = \sigma(I_j) = \frac{1}{1+e^{-I_j}}$
    - **Backpropagate the error** (by updating weights and biases)
      - For unit j in output layer: $Err_j = O_j(1 - O_j)(T_j - O_j)$
      - For unit j in a hidden layer: $Err_j = O_j(1 - O_j) \sum_k Err_k w_{jk}$
      - Update weights: $w_{ij} = w_{ij} + \eta Err_j O_i$
      - Update bias: $\theta_j = \theta_j + \eta Err_j$
- Terminating condition (when error is very small, etc.)

More on the output layer unit j

- Recall:

$$J = \frac{1}{2}\sum_j \left(T_j - O_j\right)^2, \; O_j = \sigma\left(\sum_i w_{ij}\, O_i + \theta_j\right)$$

- Chain rule of first derivation

$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial O_j}\frac{\partial O_j}{\partial w_{ij}} = -\underbrace{\left(T_j - O_j\right)O_j\left(1 - O_j\right)}_{\text{Denoted as } Err_j\text{!}}O_i$$

$$\frac{\partial J}{\partial \theta_j} = \frac{\partial J}{\partial O_j}\frac{\partial O_j}{\partial \theta_j} = -\left(T_j - O_j\right)O_j\left(1 - O_j\right)$$

More on the hidden layer unit j

- Let i, j, k denote units in input layer, hidden layer, and output layer, respectively

$$J = \frac{1}{2}\sum_k (T_k - O_k)^2, \; O_k = \sigma\left(\sum_j w_{jk} O_j + \theta_k\right), O_j = \sigma\left(\sum_i w_{ij} O_i + \theta_j\right)$$

- Chain rule of first derivation

$$\frac{\partial J}{\partial w_{ij}} = \sum_k \frac{\partial J}{\partial O_k}\frac{\partial O_k}{\partial O_j}\frac{\partial O_j}{\partial w_{ij}}$$

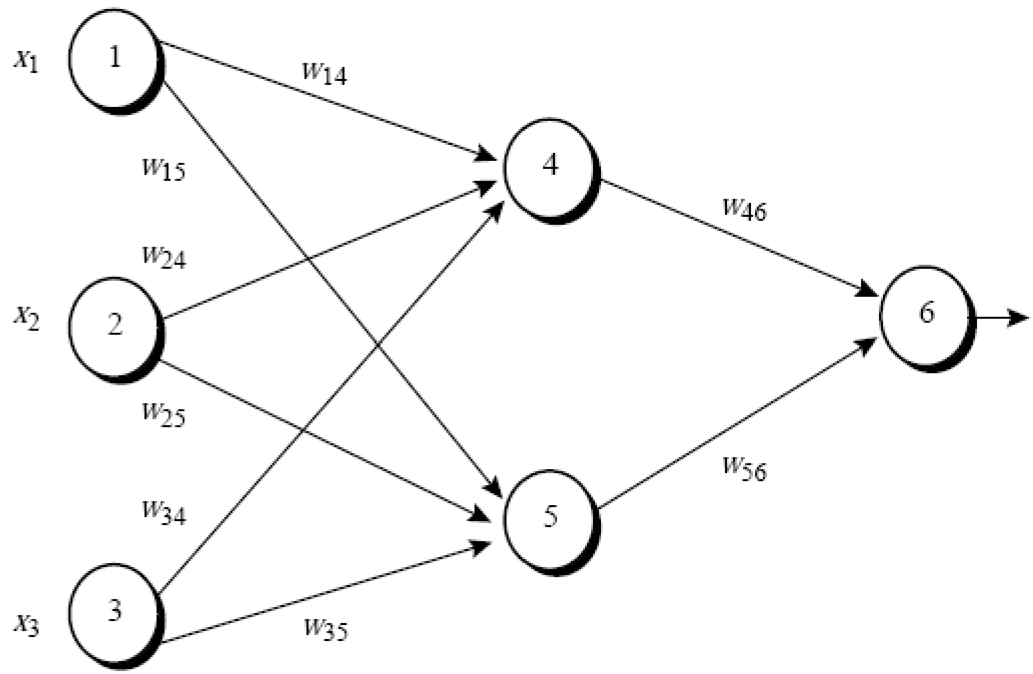
$$= -\sum_k \underbrace{\underbrace{(T_k - O_k)O_k(1-O_k)}_{Err_k}w_{jk}O_j(1-O_j)}_{Err_j}O_i$$

Err_k: **Already computed in the output layer!**

Note: $\frac{\partial J}{\partial O_k} = -(T_k - O_k), \frac{\partial O_k}{\partial O_j} = O_k(1-O_k)w_{jk}, \frac{\partial O_j}{\partial w_{ij}} = O_j(1-O_j)O_i$

$$\frac{\partial J}{\partial \theta_j} = \sum_k \frac{\partial J}{\partial O_k}\frac{\partial O_k}{\partial O_j}\frac{\partial O_j}{\partial \theta_j} = -Err_j$$

Example



A multilayer feed-forward neural network

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	-0.4	0.2	0.1

Initial Input, weight, and bias values

Example: Forward Pass

- Forward computation:

Table 9.2: The net input and output calculations.

Unit j	*Net input*, I_j	*Output*, O_j
4	$0.2 + 0 - 0.5 - 0.4 = -0.7$	$1/(1 + e^{0.7}) = 0.332$
5	$-0.3 + 0 + 0.2 + 0.2 = 0.1$	$1/(1 + e^{-0.1}) = 0.525$
6	$(-0.3)(0.332) - (0.2)(0.525) + 0.1 = -0.105$	$1/(1 + e^{0.105}) = 0.474$

Calculate net input: $\boldsymbol{I_j = \sum_i w_{ij} O_i + \theta_j}$

Calculate output of unit $\boldsymbol{j}$**:** $\boldsymbol{O_j = \sigma(I_j) = \frac{1}{1+e^{-I_j}}}$

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	-0.4	0.2	0.1

Example: backpropagation

- Error backpropagation and weight update:

Table 9.3: Calculation of the error at each node.

Unit j	Err_j
6	$(0.474)(1-0.474)(1-0.474) = 0.1311$
5	$(0.525)(1-0.525)(0.1311)(-0.2) = -0.0065$
4	$(0.332)(1-0.332)(0.1311)(-0.3) = -0.0087$

$\boldsymbol{assuming\ T_6 = 1}$

For unit j in output layer: $Err_j = O_j(1-O_j)(T_j - O_j)$

For unit j in a hidden layer: $Err_j = O_j(1-O_j)\sum_k Err_k w_{jk}$

Table 9.4: Calculations for weight and bias updating.

Weight or bias	*New value*
w_{46}	$-0.3 + (0.9)(0.1311)(0.332) = -0.261$
w_{56}	$-0.2 + (0.9)(0.1311)(0.525) = -0.138$
w_{14}	$0.2 + (0.9)(-0.0087)(1) = 0.192$
w_{15}	$-0.3 + (0.9)(-0.0065)(1) = -0.306$
w_{24}	$0.4 + (0.9)(-0.0087)(0) = 0.4$
w_{25}	$0.1 + (0.9)(-0.0065)(0) = 0.1$
w_{34}	$-0.5 + (0.9)(-0.0087)(1) = -0.508$
w_{35}	$0.2 + (0.9)(-0.0065)(1) = 0.194$
θ_6	$0.1 + (0.9)(0.1311) = 0.218$
θ_5	$0.2 + (0.9)(-0.0065) = 0.194$
θ_4	$-0.4 + (0.9)(-0.0087) = -0.408$

$\boldsymbol{assuming\ \eta = 0.9}$

Update weights: $w_{ij} = w_{ij} + \eta Err_j O_i$; **Update bias:** $\theta_j = \theta_j + \eta Err_j$

Efficiency and Interpretability

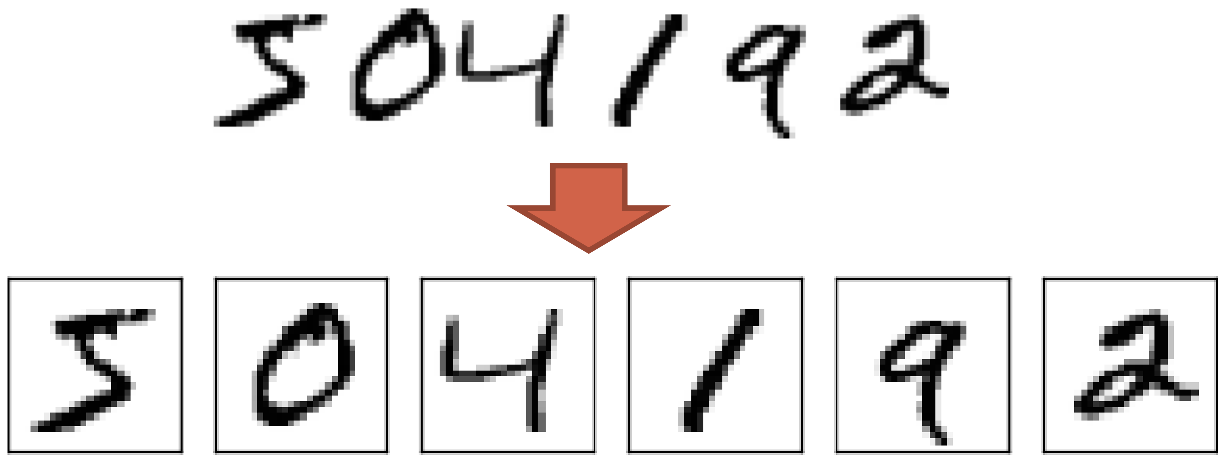
- **Efficiency** of backpropagation: Each iteration through the training set takes O(|D| * *w*), with |D| tuples and *w* weights, but # of iterations can be exponential to n, the number of inputs, in worst case
- For easier comprehension: **Rule extraction** by network pruning*
  - Simplify the network structure by removing weighted links that have the least effect on the trained network
  - Then perform link, unit, or activation value clustering
  - The set of input and activation values are studied to derive rules describing the relationship between the input and hidden unit layers
- **Sensitivity analysis**: assess the impact that a given input variable has on a network output. The knowledge gained from this analysis can be represented in rules
  - E.g., If x decreases 5% then y increases 8%

Neural Network as a Classifier

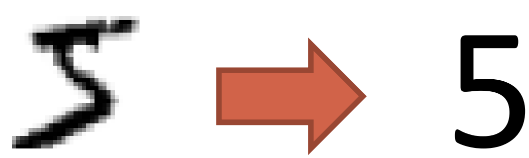
- Weakness
  - Long training time
  - Require a number of parameters typically best determined empirically, e.g., the network topology or "structure."
  - Poor interpretability: Difficult to interpret the symbolic meaning behind the learned weights and of "hidden units" in the network
- Strength
  - High tolerance to noisy data
  - Successful on an array of real-world data, e.g., hand-written letters
  - Algorithms are inherently parallel
  - Techniques have recently been developed for the extraction of rules from trained neural networks
  - Deep neural network is powerful

Digits Recognition Example

- Obtain sequence of digits by segmentation

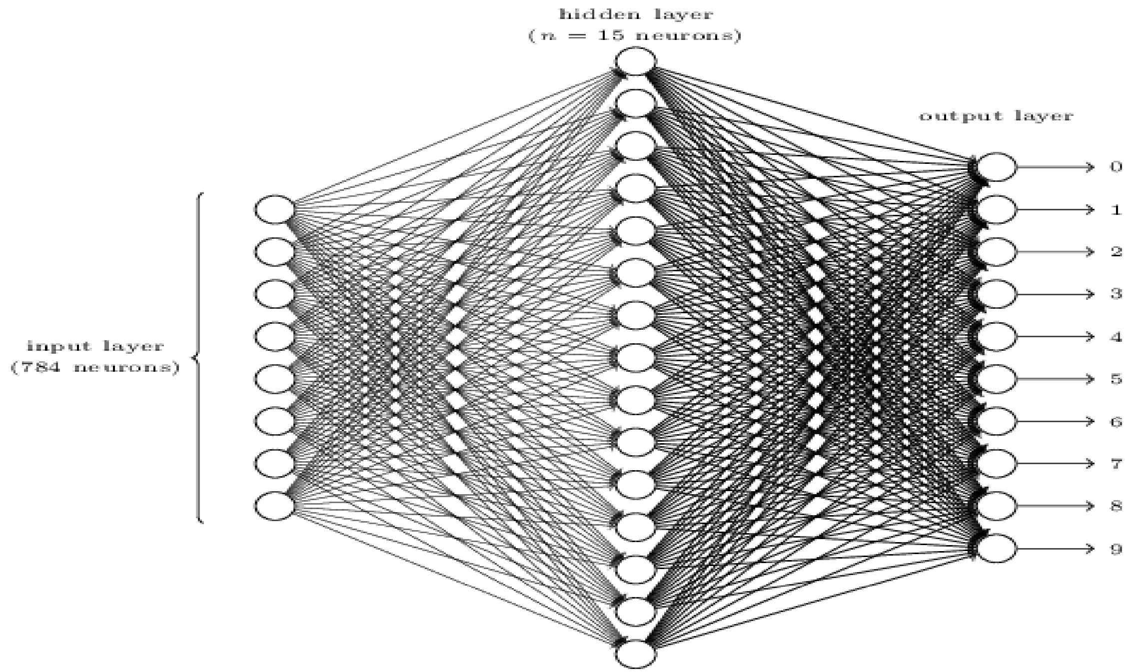


- Recognition (our focus)

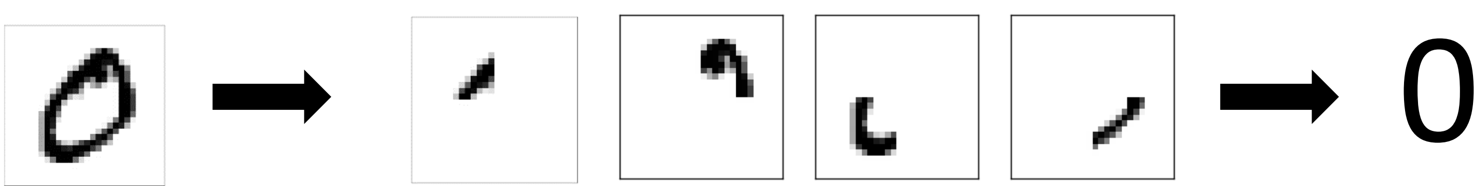


Digits Recognition Example

- The architecture of the used neural network



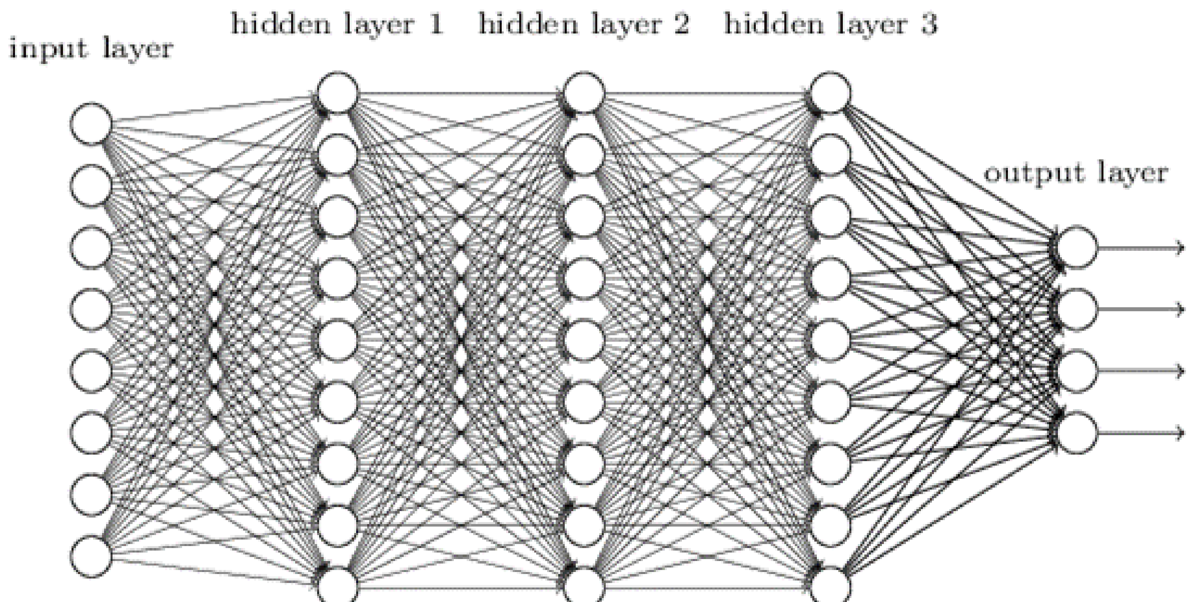
- What each neurons are doing?

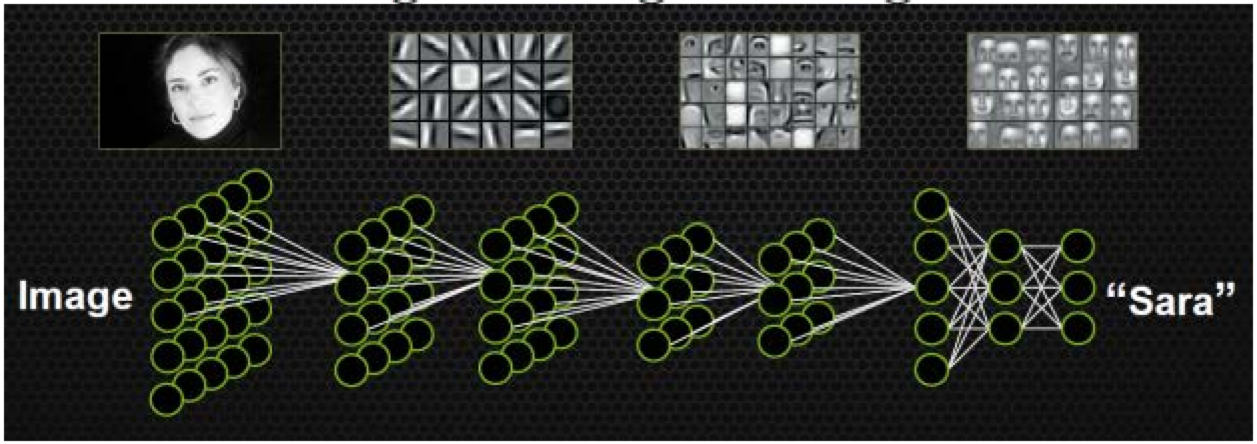


Input image | Activated neurons detecting image parts | Predicted number

Towards Deep Learning*

Deep neural network





Further References

- 3Blue1Brown NN series: https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi
- Deep Learning
  - http://neuralnetworksanddeeplearning.com/
  - http://www.deeplearningbook.org/
  - http://www.charuaggarwal.net/neural.htm

Neural Network

- Introduction
- Multi-Layer Feed-Forward Neural Network
- Summary

Summary

- Neural Network
  - Feed-forward neural networks; activation function; loss function; backpropagation