CS247: ADVANCED DATA MINING

04: Neural Network and Deep Learning

Instructor: Yizhou Sun

yzsun@cs.ucla.edu

April 16, 2019

Methods to Learn

	Vector Data	Text Data	Graph & Network	Recommender Systems
Classification	**Naïve Bayes; Logistic Regression**; **NN**		Label Propagation	
Clustering	**K-means; kernel k-means; Mixture Models**	PLSA; LDA	Spectral Clustering	Matrix Factorization
Prediction	**NN**			Collaborative Filtering; Factorization machine; Hybrid CF; Recommendation with graph regularization
Ranking			PageRank	
Similarity Search			P-PageRank	
Representation Learning		Word embedding	Network embedding	Deep collaborative learning

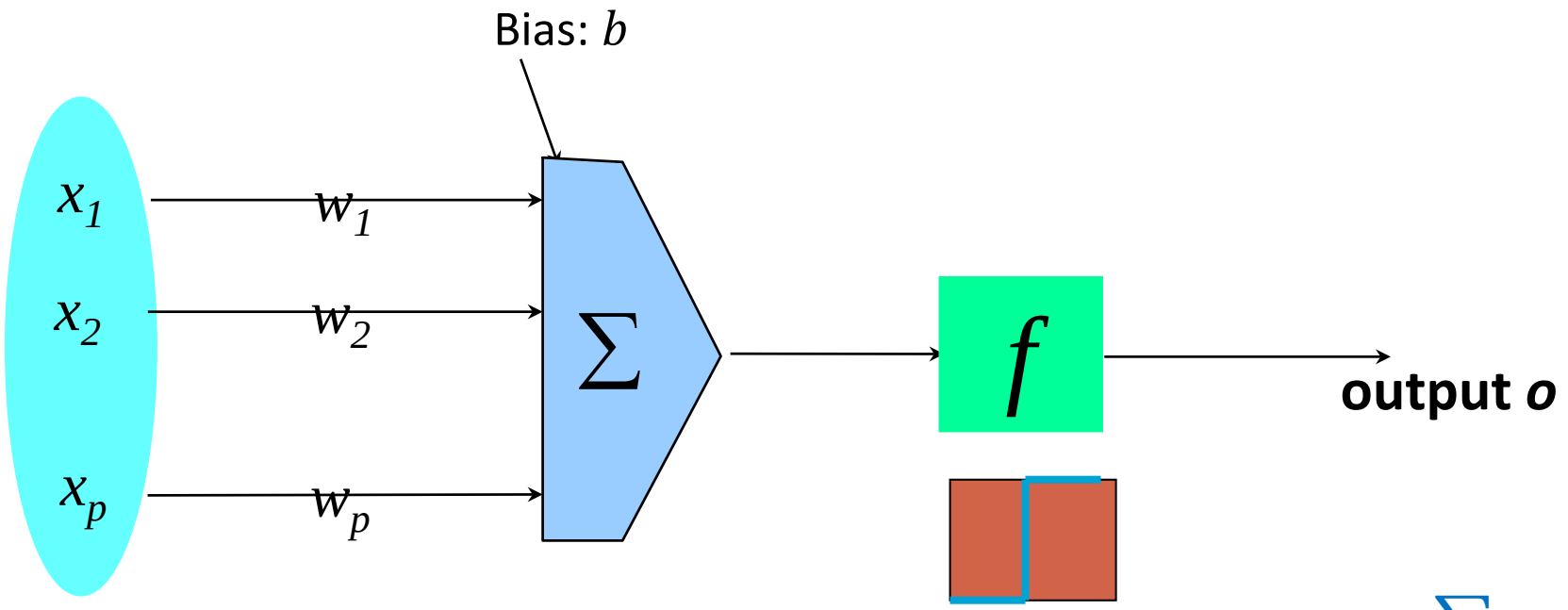
Neural Network

- Introduction 
- Connection to Shallow Machine Learning Algorithms
- Multi-Layer Feed-Forward Neural Network
- Deep Learning
- Summary

Artificial Neural Networks

- Consider humans:
  - Neuron switching time ~.001 second
  - Number of neurons $\sim 10^{10}$
  - Connections per neuron $\sim 10^{4-5}$
  - 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



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

Input vector x **weight vector w** **weighted sum** **Activation function**

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

Important Concepts

- Architecture
- Activation function
- Loss function
- Optimization
- Regularization

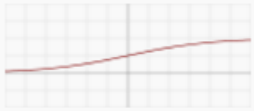
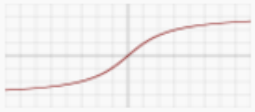
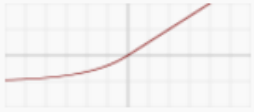
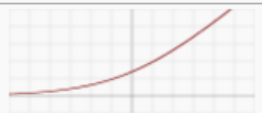
Architecture

- Decide the **network topology:**
  - Specify # of units in the *input layer*, # of *hidden layers* (if > 1), # of units in *each hidden layer*, the unit types, connection between layers, and # of units in the *output layer*
- Architecture specifies the function that maps input to output, which contains parameters to be learned

Activation function

- An activation function $f(\cdot)$ in the output layer can control the nature of the output (e.g., probability value in [0, 1])
- Activation functions bring nonlinearity into hidden layers, which increases the complexity of the model.
- Good activation functions should be differentiable for optimization purpose

Examples of 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		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Loss Functions

- How good are the outputs compared with the labels?
  - Empirical risk
    - $\mathcal{L}(\boldsymbol{w}) = \frac{1}{n}\sum_i l\,(y^{(i)}, \hat{y}^{(i)})$, where $\hat{y}^{(i)} = f\,(\boldsymbol{x}^{(i)}, \boldsymbol{w})$
    - $\boldsymbol{w}$: parameters in the model
  - Loss function: difference between actual value and predicted value
    - $l(y, \hat{y})$

Example of Loss Functions

- Squared error
  - $l(y, \hat{y}) = (y - \hat{y})^2$
- (Binary) cross entropy loss
  - $l(y, \hat{y}) = -y log \hat{y} - (1 - y) log(1 - \hat{y})$
  - $y \in \{0,1\}, \hat{y} \in [0,1]$
- Hinge loss
  - $l(y, \hat{y}) = \max(0, 1 - y\hat{y})$
  - $y \in \{-1,1\}$

Optimization

- Given a training dataset, minimize the empirical risk
  - Find $\boldsymbol{w}$, such that $\mathcal{L}(\boldsymbol{w}) = \frac{1}{n}\sum_i l\left(y^{(i)}, \hat{y}^{(i)}\right)$ is minimized
- Solution:
  - Stochastic gradient descent + chain rule = backpropagation

Regularization

- Avoid overfitting
- Techniques
  - L2/L1 regularization
  - Dropout
  - Early stopping
  - …

Neural Network

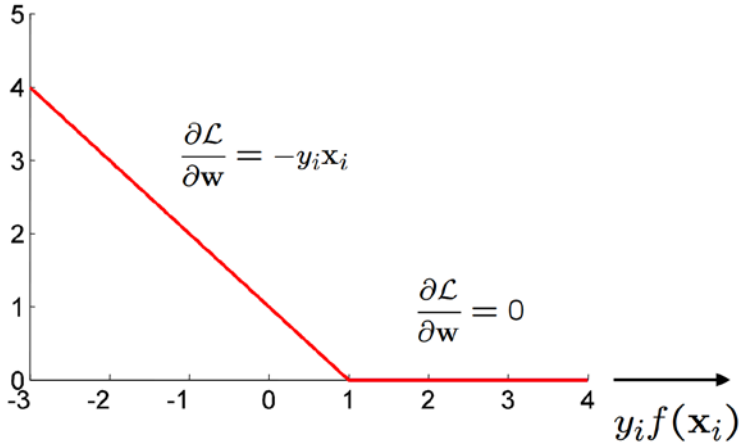
- Introduction
- Connection to Shallow Machine Learning Algorithms 
- Multi-Layer Feed-Forward Neural Network
- Deep Learning
- Summary

Perceptron

- Architecture:
  - A single neuron
- Activation function
  - Training: identify function
  - Inference: sign function/step function
- Loss function
  - $l(y, \hat{y}) = \max(0, -y\hat{y})$
- Optimization
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta y^{(i)} \boldsymbol{x}^{(i)}$, for a misclassified training data point $\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)$, i.e., $y^{(i)} w^T \boldsymbol{x}^{(i)} < 0$
  - η: learning rate

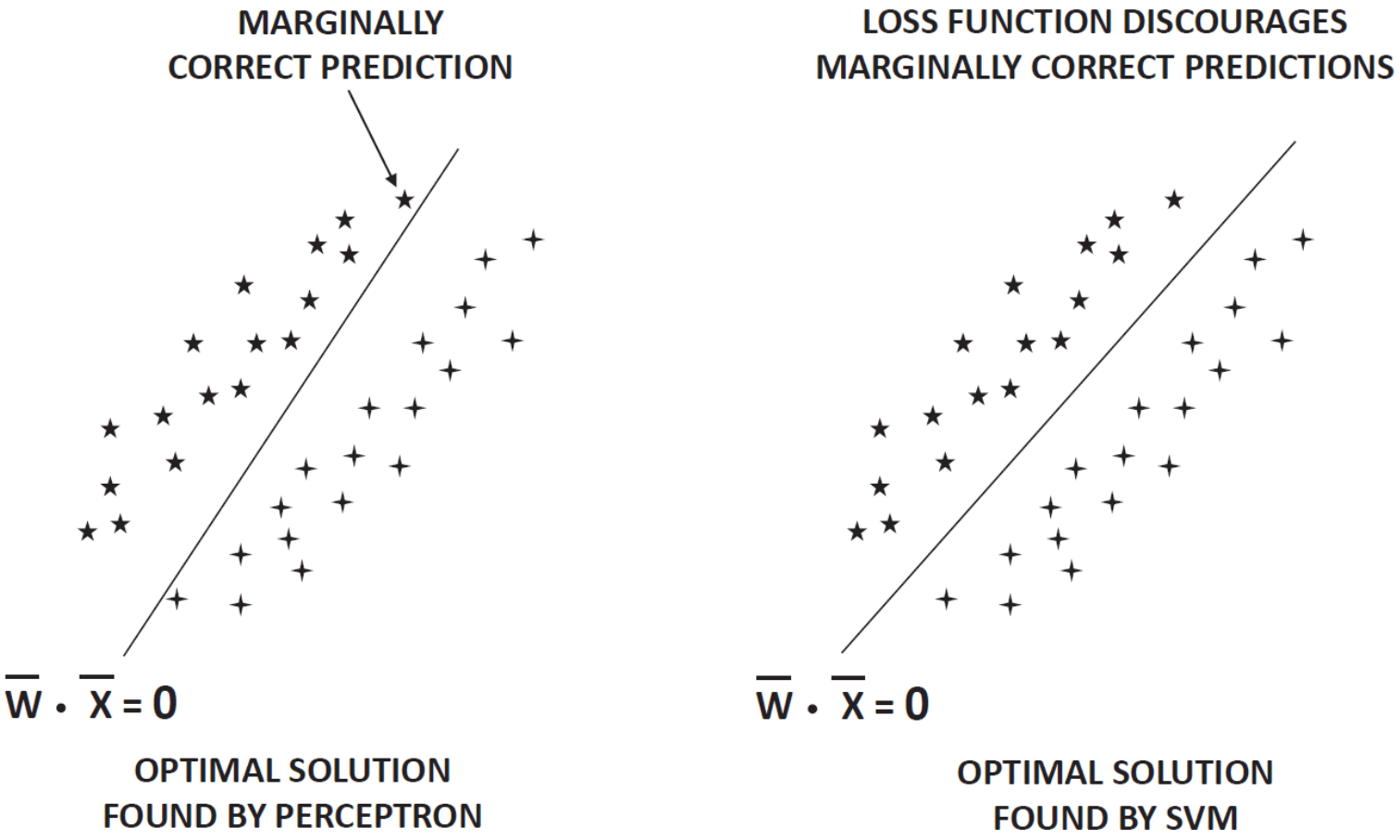
Linear SVM

- Architecture:
  - A single neuron
- Activation function
  - Training: identify function
  - Inference: sign function/step function
- Loss function
  - $l(y, \hat{y}) = \max(0, 1 - y\hat{y})$
- Regularization
  - L2, i.e., $\frac{1}{2}\lambda||\boldsymbol{w}||^2$
- Optimization
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta(\lambda \boldsymbol{w} - y^{(i)}\boldsymbol{x}^{(i)})$, for a misclassified or barely correct training data point $(\boldsymbol{x}^{(i)}, y^{(i)})$, i.e., $y^{(i)}w^T\boldsymbol{x}^{(i)} < 1$
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta\lambda\boldsymbol{w}$, for a confidently correct training data point
  - η: learning rate



Perceptron V.S. Linear SVM

- Source: http://www.charuaggarwal.net/neural.htm



Logistic Regression

- Architecture:
  - A single neuron
- Activation function
  - Sigmoid function
- Loss function
  - $l(y, \hat{y}) = -y log \hat{y} - (1 - y) log(1 - \hat{y})$
  - Note $\hat{y}$ is the predicted probability of taking class 1
- Optimization
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta(y^{(i)} - \sigma(w^T \boldsymbol{x}^{(i)}))\boldsymbol{x}^{(i)}$, for a training data point $\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)$
  - η: learning rate

Question

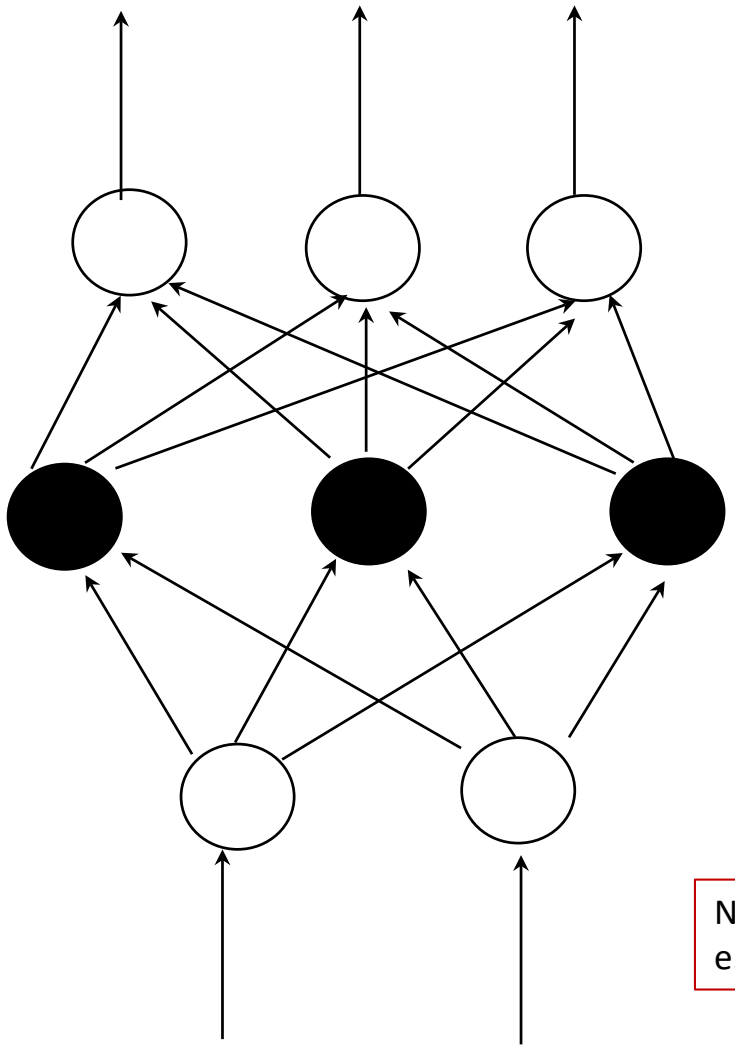
- How about multivariate logistic regression?

Neural Network

- Introduction
- Connection to Shallow Machine Learning Algorithms
- Multi-Layer Feed-Forward Neural Network
- Deep Learning
- Summary

A Multi-Layer Feed-Forward Neural Network

A **two-layer** network



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

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

Bias term

Weight matrix

Nonlinear transformation,
e.g. sigmoid transformation

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**

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

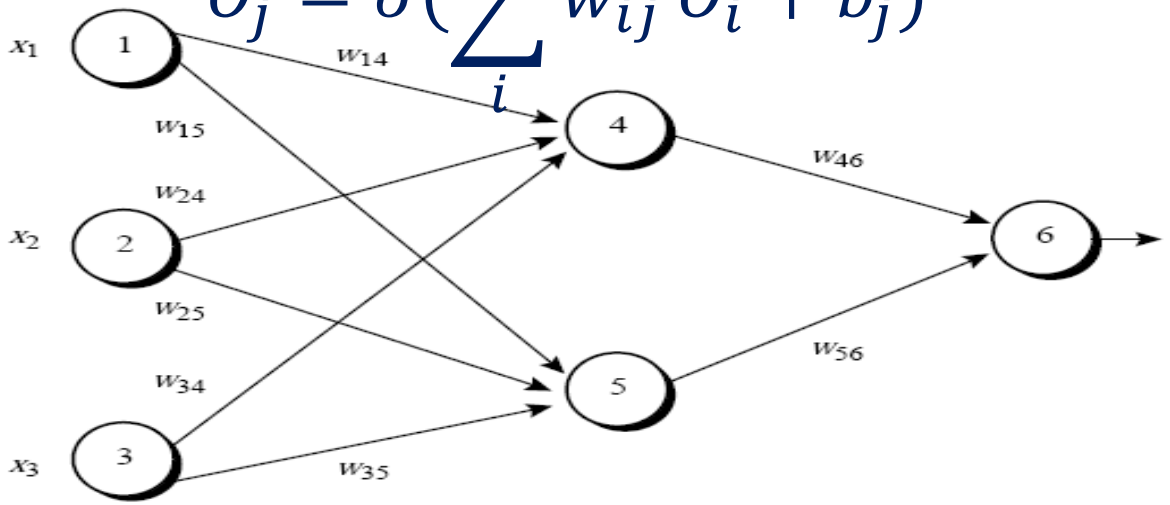
Backpropagation

- 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**"

A Toy Example

- Activation function: Sigmoid

$$O_j = \sigma(\sum_i w_{ij} O_i + b_j)$$



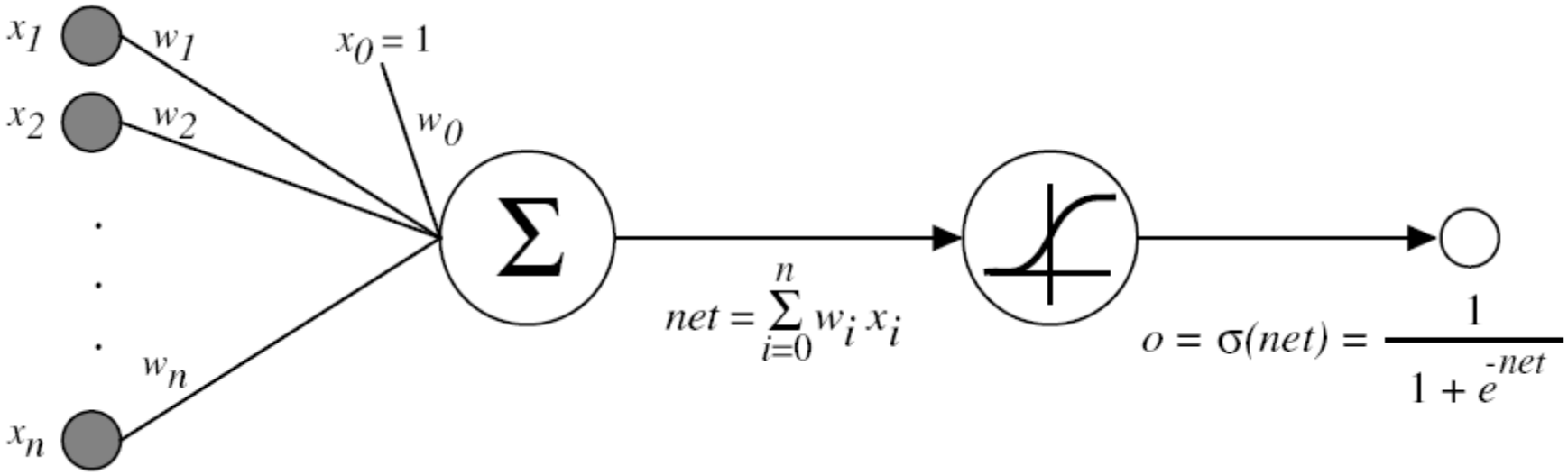
- Loss function: squared error loss

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

T_j: *true value of output unit j*;

O_j: *output value*

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

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: $I_j = \sum_i w_{ij} O_i + b_j$
        - 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 each unit j in output layer: $Err_j = O_j(1 - O_j)(T_j - O_j)$
      - For each 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: $b_j = b_j + \eta Err_j$
- Terminating condition (when error is very small, etc.)

More on the output layer unit j

- Recall:

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

- Chain rule of first derivation

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

$$\frac{\partial J}{\partial b_j} = \frac{\partial J}{\partial O_j}\frac{\partial O_j}{\partial b_j} = -(T_j - O_j)O_j(1 - O_j)$$

More on the hidden layer unit j

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

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

- Chain rule of first derivation

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

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

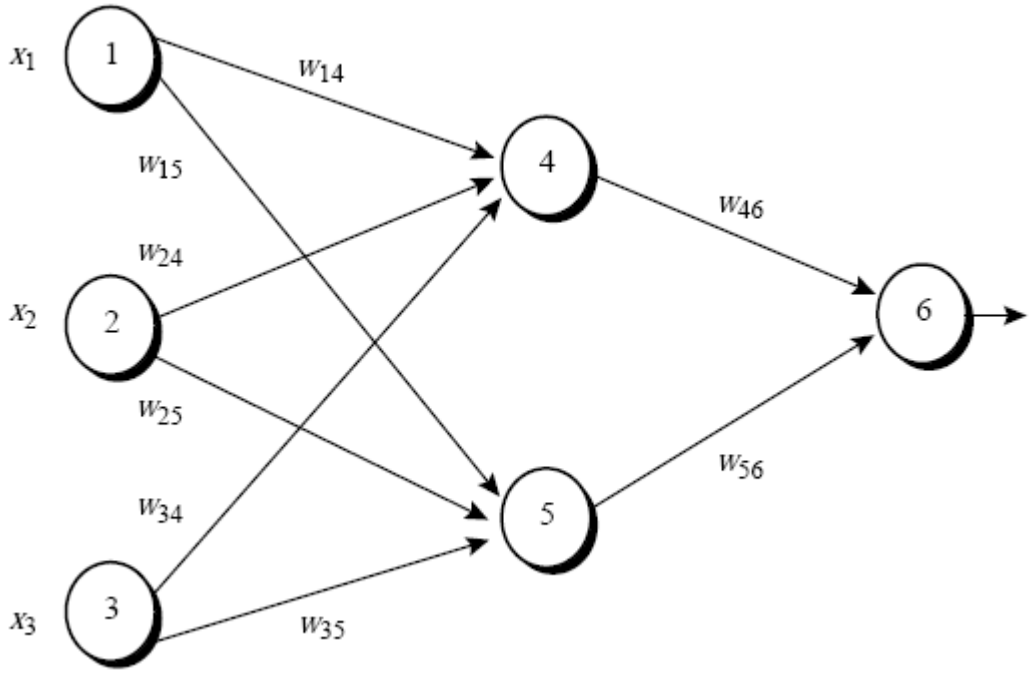
Err_k: Already computed in the output layer!

Err_j

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

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

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}	b_4	b_5	b_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: $\boldsymbol{I_j = \sum_i w_{ij} O_i + b_j}$

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}	b_4	b_5	b_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	$(0.332)(1-0.332)(0.1311)(-0.3) = -0.0087$

$assuming\ T_6 = 1$

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}	$0.2 + (0.9)(-0.0065)(1) = 0.194$
b_4	$0.1 + (0.9)(0.1311) = 0.218$
b_5	$0.2 + (0.9)(-0.0065) = 0.194$
b_6	$-0.4 + (0.9)(-0.0087) = -0.408$

$assuming\ \eta = 0.9$

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

Efficiency and Interpretability

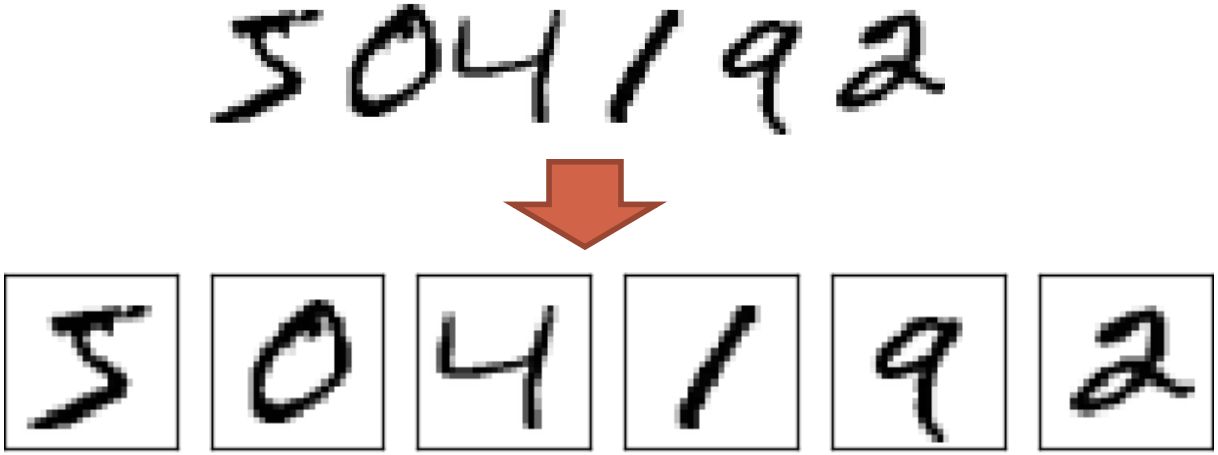
- **Efficiency** of backpropagation: Each iteration through the training set takes O(|D| * *w*), with |D| tuples and *w* weights, but # of iterations can be exponential to n, the number of inputs, in worst case
- For easier comprehension: **Rule extraction** by network pruning*
  - Simplify the network 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

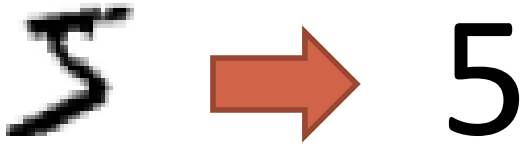
- Weakness
  - Long training time
  - Require a number of hyper-parameters typically best determined empirically, e.g., the network topology or "structure."
  - Poor interpretability: Difficult to interpret the symbolic meaning behind the learned weights and of "hidden units" in the network
- Strength
  - 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

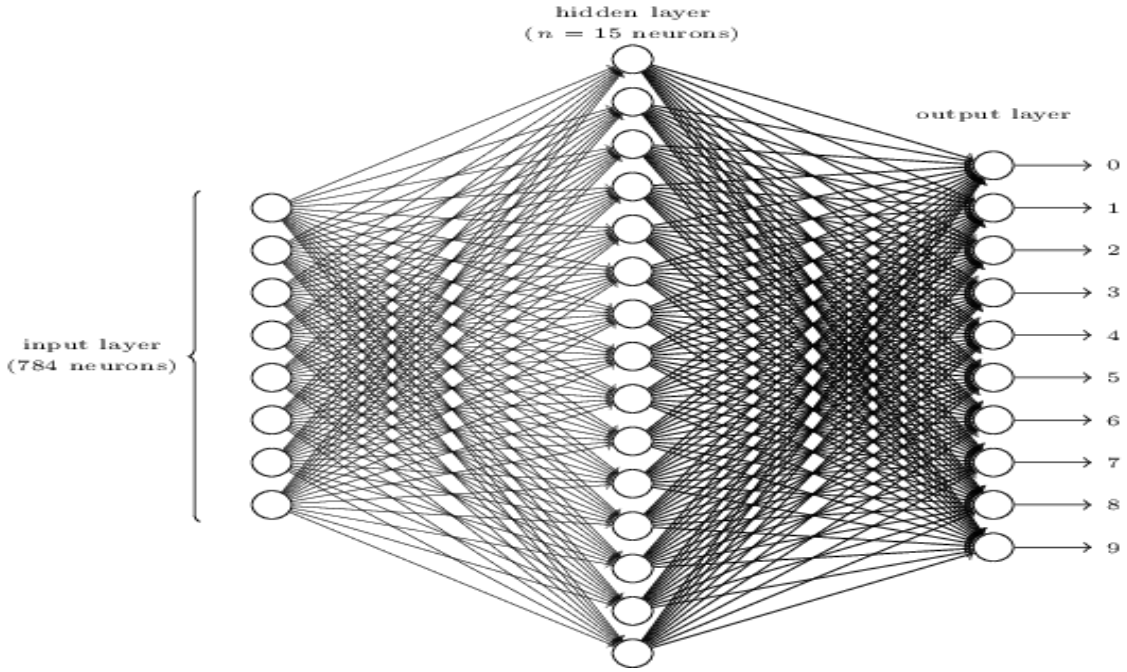


- Recognition (our focus)

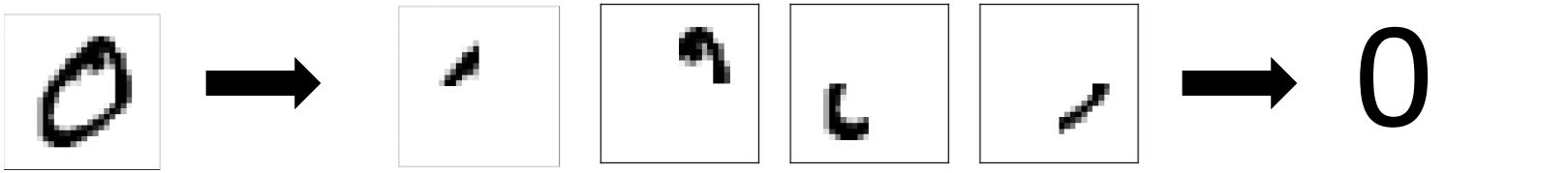


Digits Recognition Example

- The architecture of the used neural network



- What each neurons are doing?



Input image

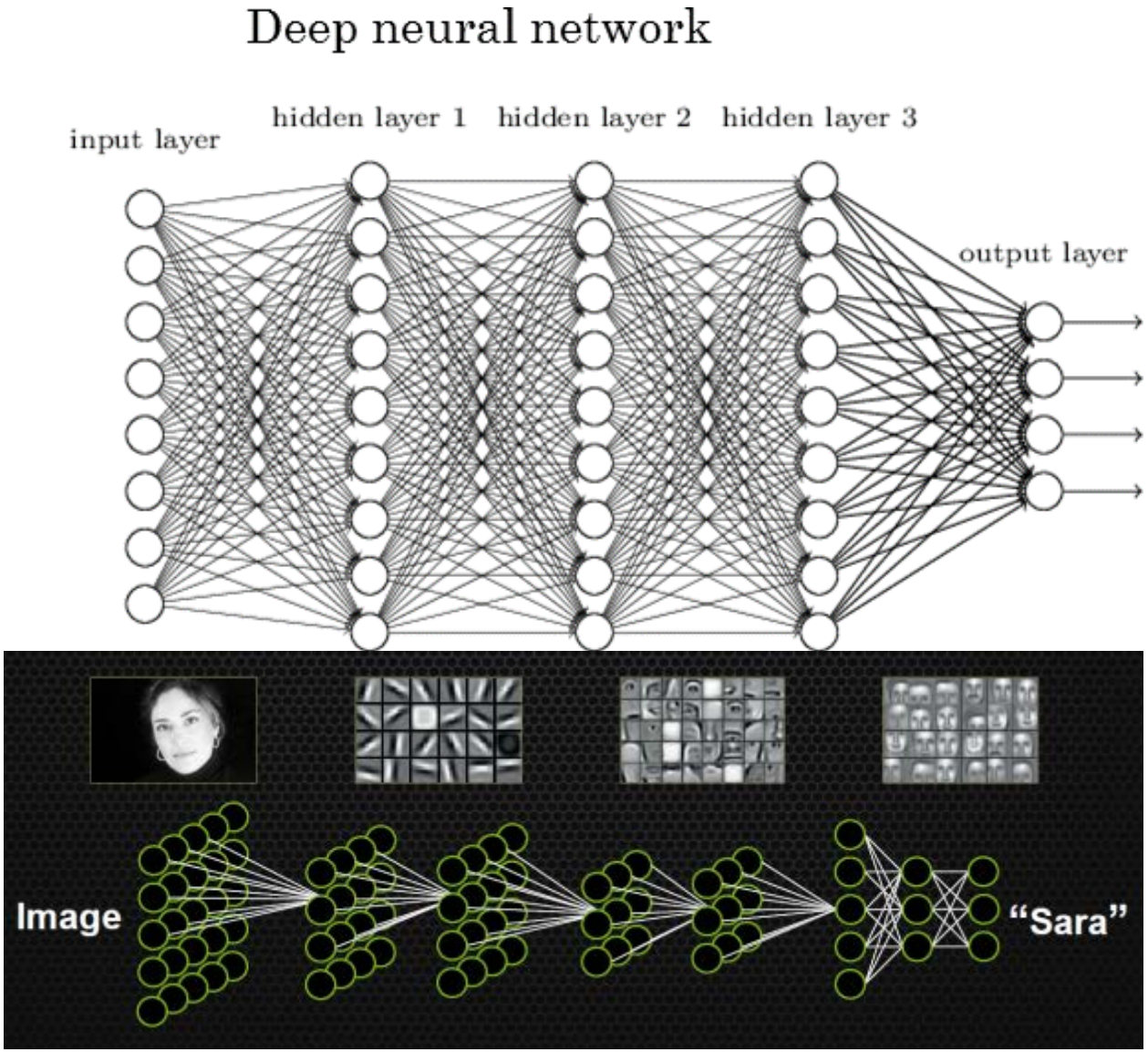
Activated neurons detecting image parts

Predicted number

Neural Network

- Introduction
- Connection to Shallow Machine Learning Algorithms
- Multi-Layer Feed-Forward Neural Network
- Deep Learning 
- Summary

Towards Deep Learning

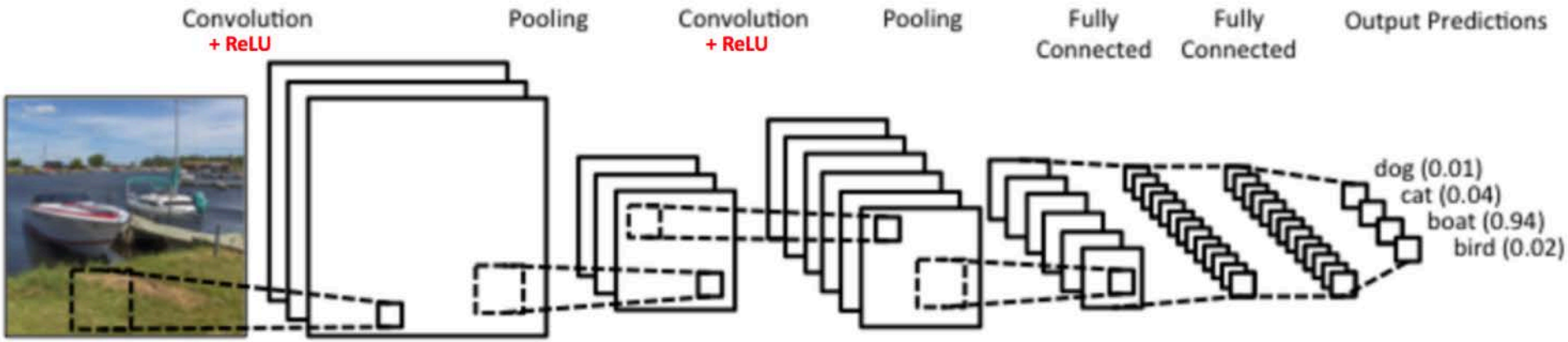


Popular Deep NNs

- Deep layers and parameter sharing
  - Convolutional Neural Network
    - Applications to: Images, NLP, Graph
  - Recurrent Neural Network
    - Applications to: Sequence data

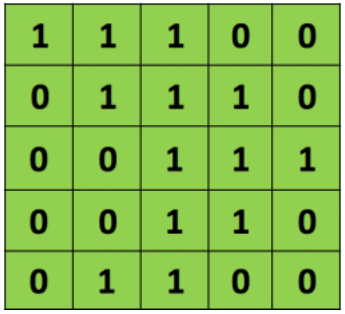
Convolutional Neural Network

- The LeNet Architecture (By Yann LeCun et al.)



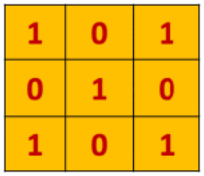
Source: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

Convolution



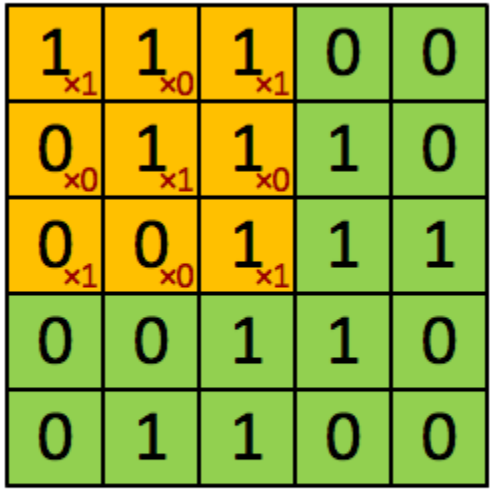
Input

*

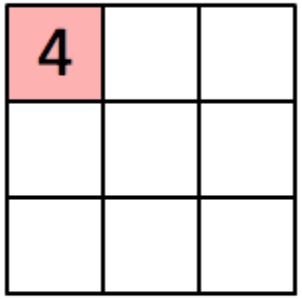


Filter

Convolution operator: weighted sum where weights are from filter matrix

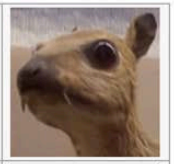
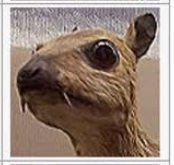
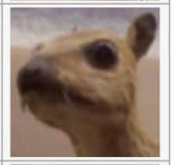
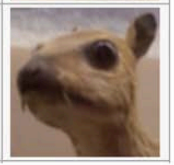


Image



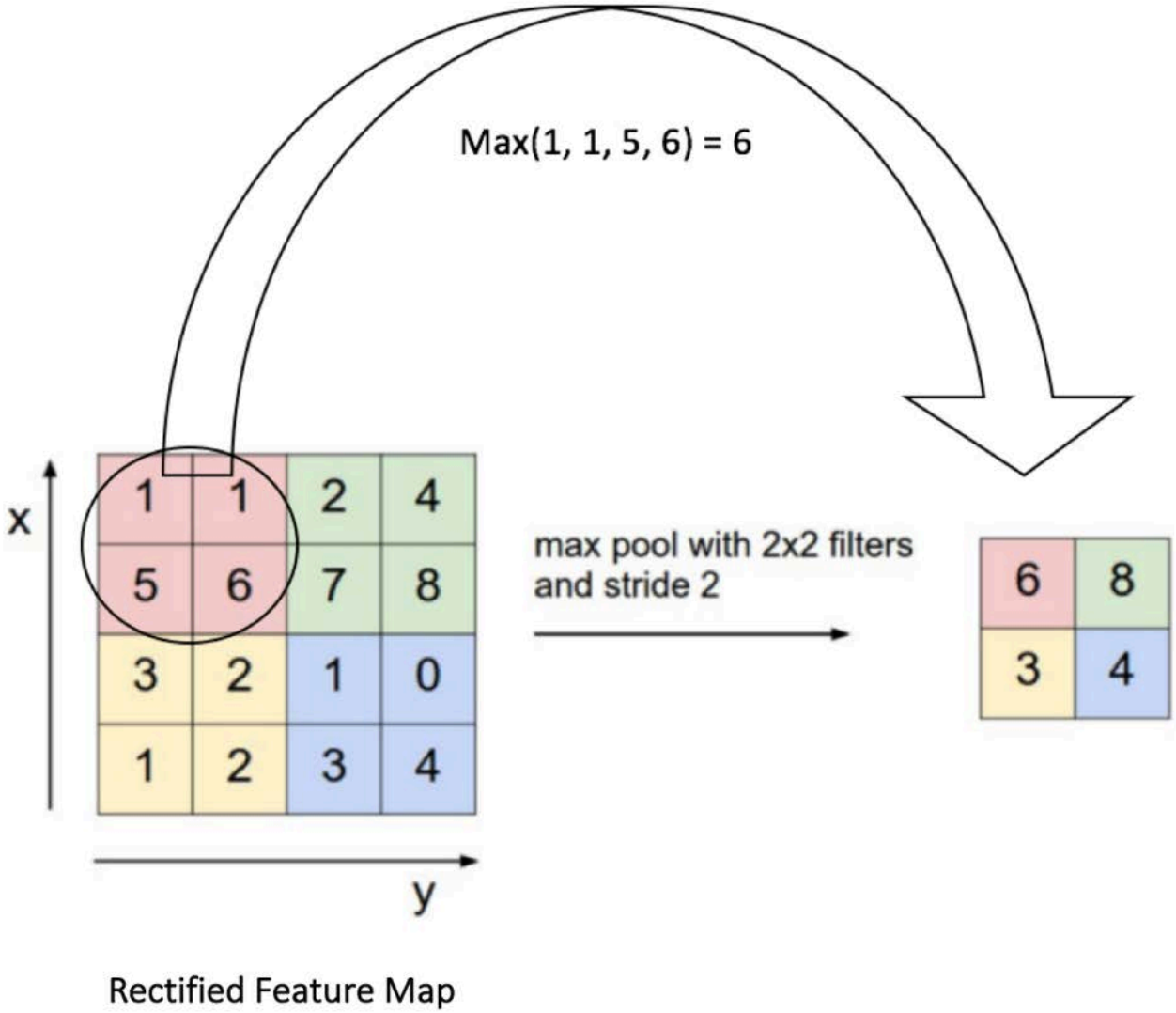
Convolved Feature

Effects of Different Filters

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Source: https://en.wikipedia.org/wiki/Kernel_(image_processing)

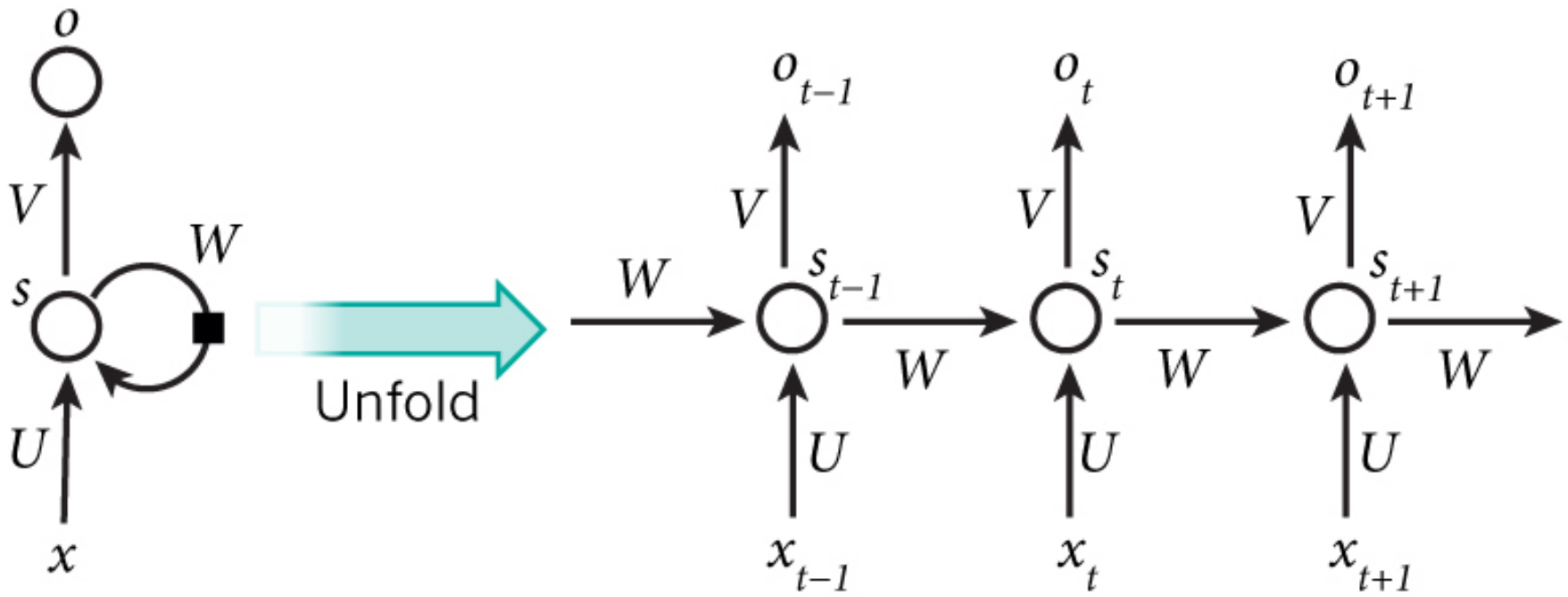
Pooling



Source: http://cs231n.github.io/convolutional-networks/

Recurrent Neural Network

- Model sequential information

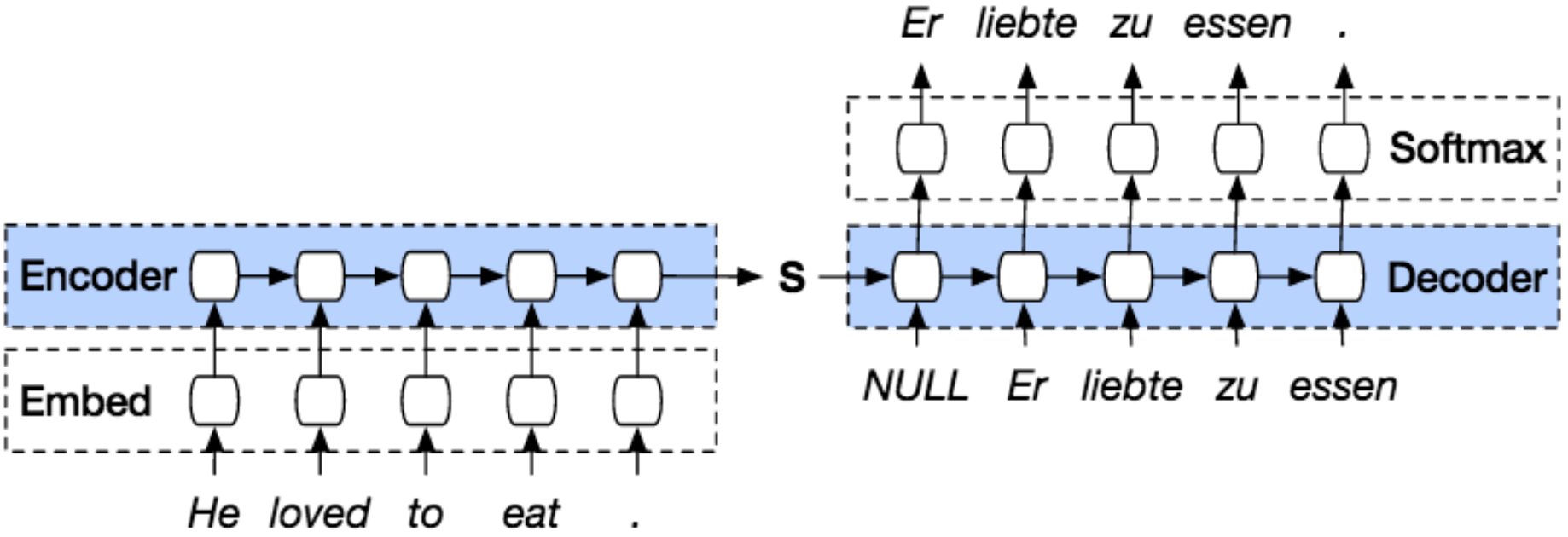


x_t: *input vector at time t*
s_t: *hidden state at time t*
o_t: *output vector at time t*

$$s_t = f(Ws_{t-1} + Ux_t)$$
$$o_t = g(Vs_t)$$

Application of Machine Translation

- Seq2Seq model



Neural Network

- Introduction
- Connection to Shallow Machine Learning Algorithms
- Multi-Layer Feed-Forward Neural Network
- Deep Learning
- Summary 

Summary

- Neural Network
  - Architecture; activation function; loss function; backpropagation
- Existing shallow machine learning algorithms can be represented in NN
- Multilayer feedforward NN
- Deep learning

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
  - http://d2l.ai/index.html