

CS247: ADVANCED DATA MINING

04: Neural Network and Deep Learning

Instructor: Yizhou Sun


yzsun@cs.ucla.edu

April 16, 2020

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

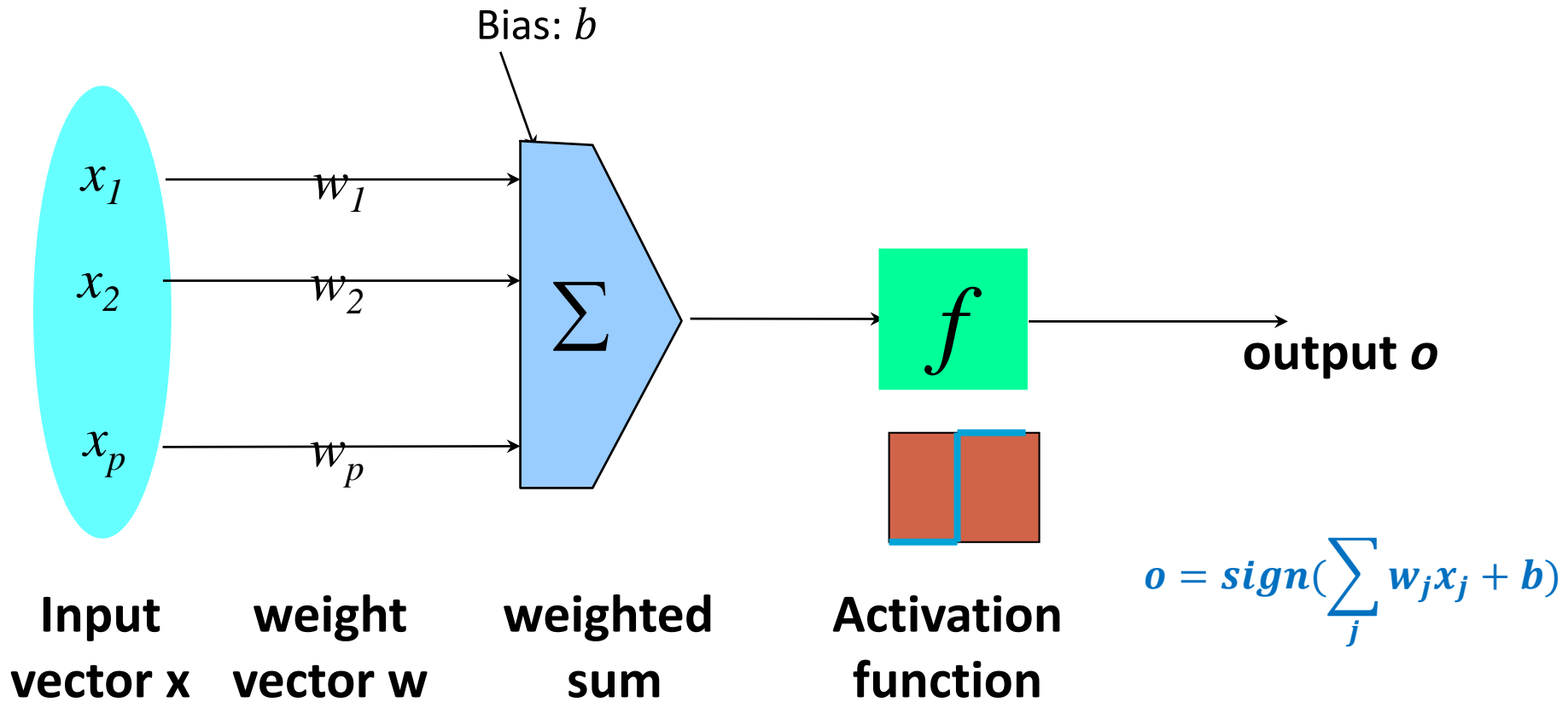
Neural Network

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

Artificial Neural Networks

- Consider humans:
 - Number of neurons $\sim 10^{10}$
 - Connections per neuron $\sim 10^{4-5}$
 - Neuron switching time $\sim .001$ second
 - Scene recognition time $\sim .1$ 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



Important Concepts

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



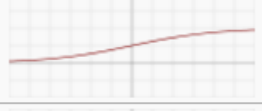
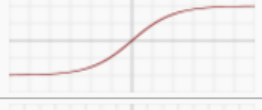

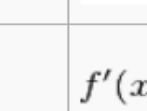

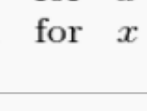
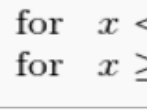
Architecture

- 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 (target)?
 - Empirical risk
 - $\mathcal{L}(\mathbf{w}) = \frac{1}{n} \sum_i l(y^{(i)}, \hat{y}^{(i)})$, where $\hat{y}^{(i)} = f(\mathbf{x}^{(i)}, \mathbf{w})$
 - \mathbf{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
 - $l(y, \hat{y}) = \max(0, 1 - y\hat{y})$
 - $y \in \{-1, 1\}, \hat{y} \in (-\infty, +\infty)$


Optimization

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

Regularization

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

Neural Network

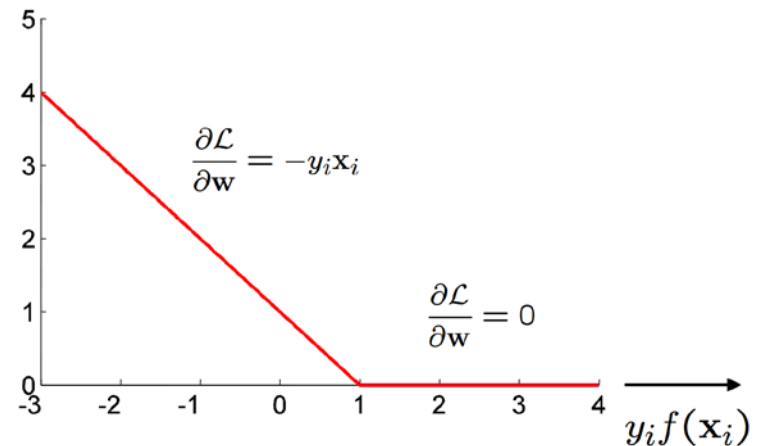
- Introduction
- Connection to Shallow Machine Learning Algorithms 
- Multi-Layer Feed-Forward Neural Network
- Deep Learning
- Summary

Perceptron

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

Linear SVM

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

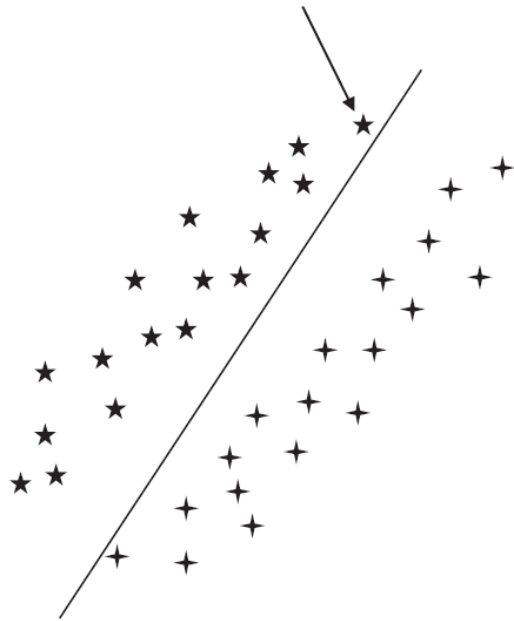


Perceptron V.S. Linear SVM

- Source:

<http://www.charuaggarwal.net/neural.htm>

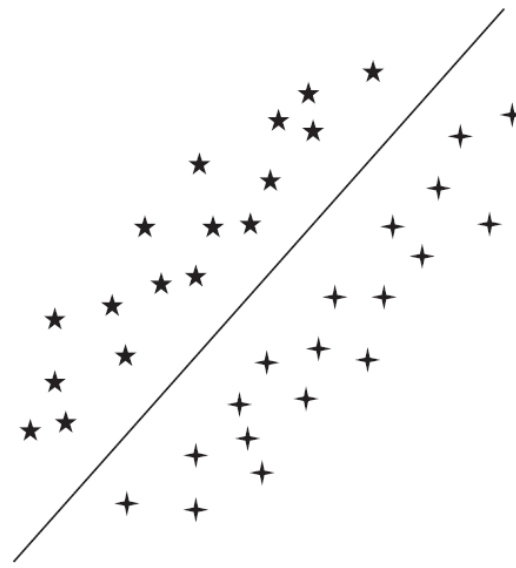
MARGINALLY
CORRECT PREDICTION



$$\bar{W} \cdot \bar{X} = 0$$

OPTIMAL SOLUTION
FOUND BY PERCEPTRON

LOSS FUNCTION DISCOURAGES
MARGINALLY CORRECT PREDICTIONS



$$\bar{W} \cdot \bar{X} = 0$$

OPTIMAL SOLUTION
FOUND BY SVM


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
 - $\mathbf{w} \leftarrow \mathbf{w} + \eta (y^{(i)} - \sigma(\mathbf{w}^T \mathbf{x}^{(i)})) \mathbf{x}^{(i)}$, for a training data point $(\mathbf{x}^{(i)}, y^{(i)})$
 - η : learning rate

Question

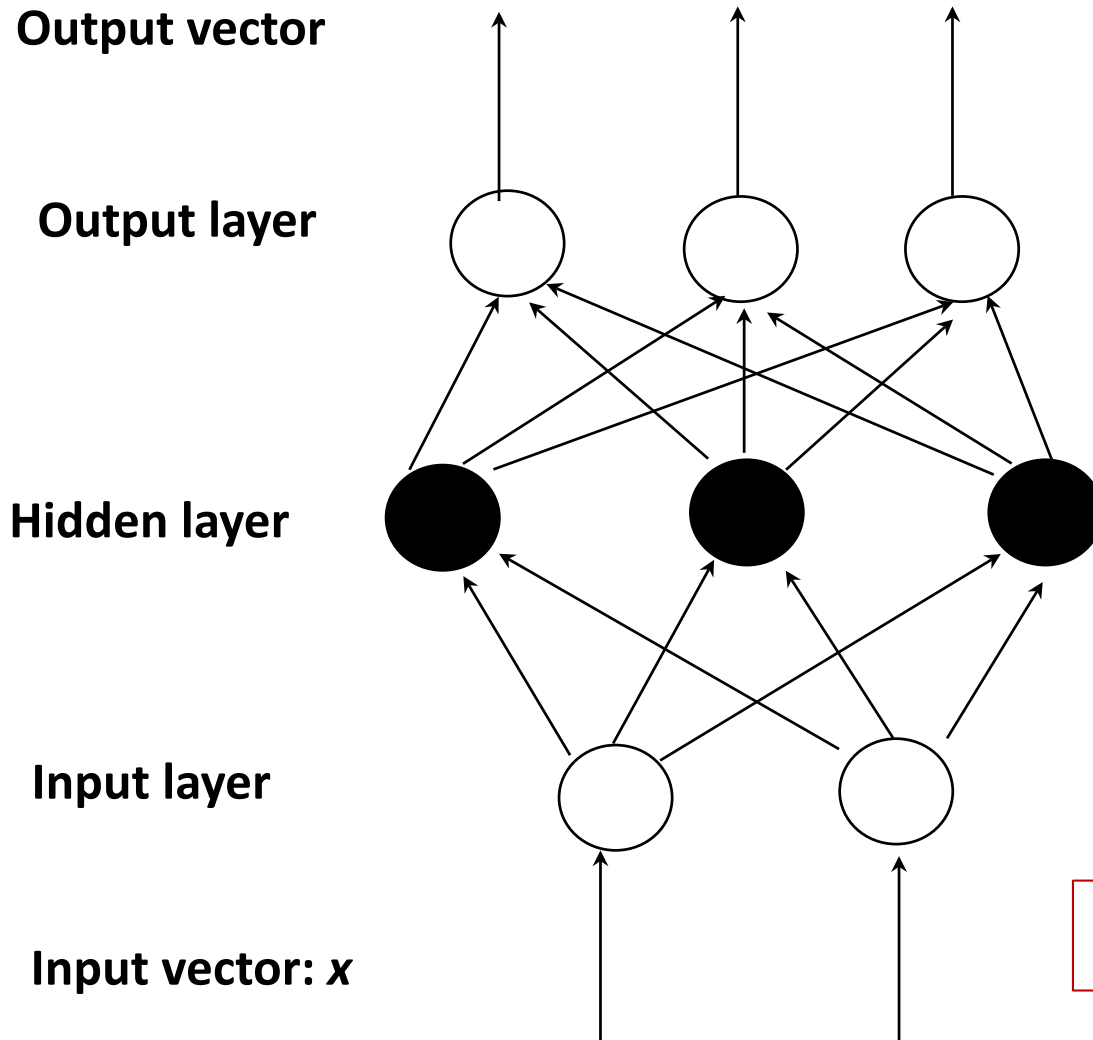
- How about multivariate logistic regression?

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



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

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

Bias term

Weight matrix

Nonlinear transformation,
e.g. sigmoid transformation

How A Multi-Layer Neural Network Works

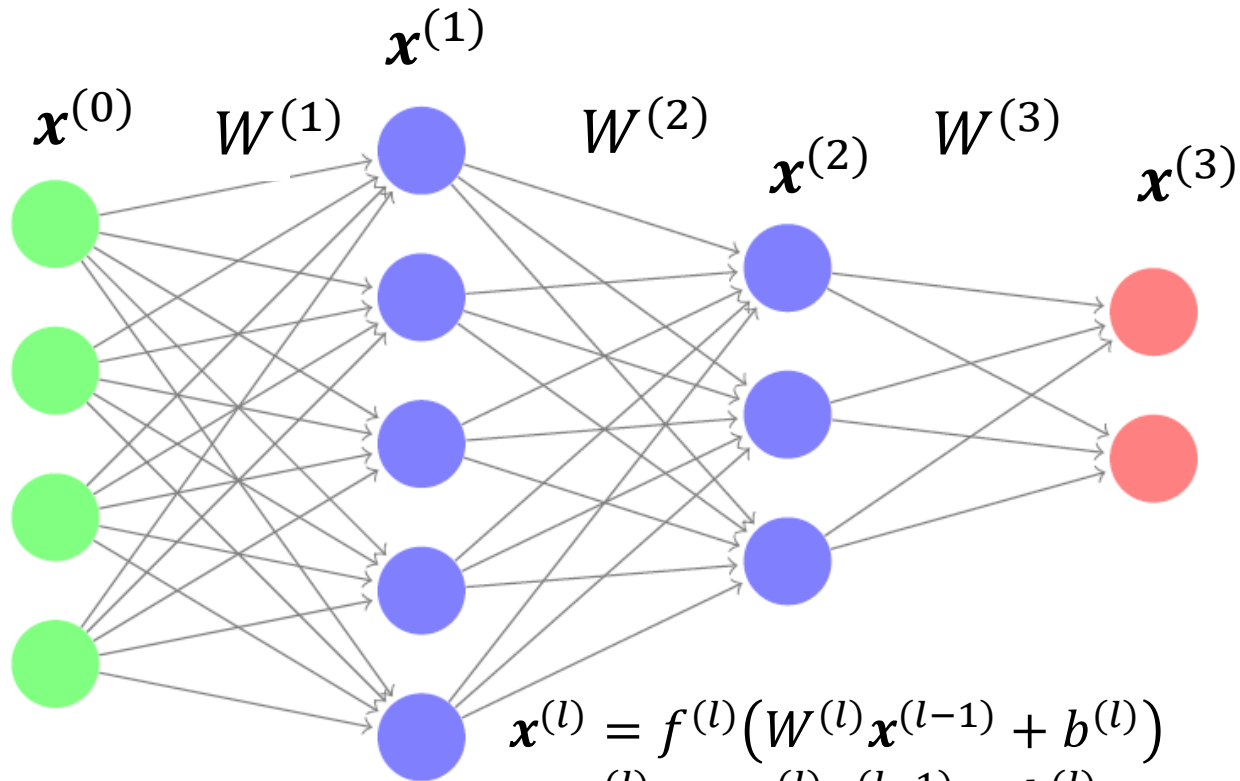
- The **inputs** to the network correspond to the attributes measured for each training tuple
- Inputs are fed simultaneously into the units making up the **input layer**
- They are then weighted and fed simultaneously to a **hidden layer**
 - The number of hidden layers is arbitrary
- The weighted outputs 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

- 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

- Loss function: $E = \frac{1}{2} \|\mathbf{y} - \hat{\mathbf{y}}\|^2$



$$\mathbf{x}^{(l)} = f^{(l)}(W^{(l)}\mathbf{x}^{(l-1)} + b^{(l)})$$

$$\mathbf{z}^{(l)} = W^{(l)}\mathbf{x}^{(l-1)} + b^{(l)}$$

$$E = \frac{1}{2} \|\mathbf{y} - \mathbf{x}^{(3)}\|^2$$

Gradient for Layer 3 (Last Layer)

- Stochastic gradient for $W_{ij}^{(3)}$ and $b_i^{(3)}$

- *Recall:*

- $E = \frac{1}{2} \|y - \mathbf{x}_3\|^2 = \frac{1}{2} \sum_i (y_i - x_i^{(3)})^2$

- $x_i^{(3)} = f^{(3)}(z_i^{(3)})$

- $z_i^{(3)} = \sum_j W_{ij} x_j^{(2)} + b_i^{(3)}$

- $$\frac{\partial E}{\partial W_{ij}^{(3)}} = \frac{\partial E}{\partial x_i^{(3)}} \frac{\partial x_i^{(3)}}{\partial z_i^{(3)}} \frac{\partial z_i^{(3)}}{\partial W_{ij}^{(3)}} = \underbrace{-(y_i - x_i^{(3)}) f'^{(3)}(z_i^{(3)})}_{\delta_i^{(3)}} x_j^{(2)}$$

- $$\frac{\partial E}{\partial b_i^{(3)}} = \frac{\partial E}{\partial x_i^{(3)}} \frac{\partial x_i^{(3)}}{\partial z_i^{(3)}} \frac{\partial z_i^{(3)}}{\partial b_i^{(3)}} = -(y_i - x_i^{(3)}) \delta_i^{(3)} f'^{(3)}(z_i^{(3)})$$

Gradient for Layer 2

- Stochastic gradient for $W_{ij}^{(2)}$ and $b_i^{(2)}$

- Recall:*

- $$E = \frac{1}{2} \|y - \mathbf{x}_3\|^2 = \frac{1}{2} \sum_i (y_i - x_i^{(3)})^2; x_i^{(3)} = f^{(3)}(z_i^{(3)}); z_i^{(3)} = \sum_j W_{ij} x_j^{(2)} + b_i^{(3)}$$

- $$x_j^{(2)} = f^{(2)}(z_j^{(2)}); z_j^{(2)} = \sum_k W_{jk} x_k^{(1)} + b_j^{(2)}$$

- $$\frac{\partial E}{\partial W_{jk}^{(2)}} = \sum_i \frac{\partial E}{\partial x_i^{(3)}} \frac{\partial x_i^{(3)}}{\partial z_i^{(3)}} \frac{\partial z_i^{(3)}}{\partial x_j^{(2)}} \frac{\partial x_j^{(2)}}{\partial z_j^{(2)}} \frac{\partial z_j^{(2)}}{\partial W_{jk}^{(2)}}$$

$$= \sum_i \underbrace{-(y_i - x_i^{(3)}) f'^{(3)}(z_i^{(3)})}_{\delta_i^{(3)}} \underbrace{W_{ij}^{(2)} f'^{(2)}(z_j^{(2)})}_{\delta_j^{(2)}} x_k^{(1)}$$

Gradient for Layer 1

- Stochastic gradient for $W_{ks}^{(1)}$

- $$\frac{\partial E}{\partial W_{ks}^{(1)}} = \sum_j \delta_j^{(2)} W_{jk}^{(2)} f'^{(1)} \left(z_k^{(1)} \right) x_s^{(0)}$$

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

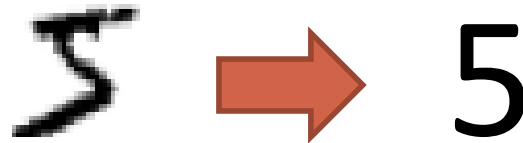
- High tolerance to noisy data
- Successful on an array of real-world data, e.g., hand-written letters
- Algorithms are inherently parallel
- Techniques have recently been developed for the extraction of rules from trained neural networks
- Deep neural network is powerful

Digits Recognition Example

- Obtain sequence of digits by segmentation

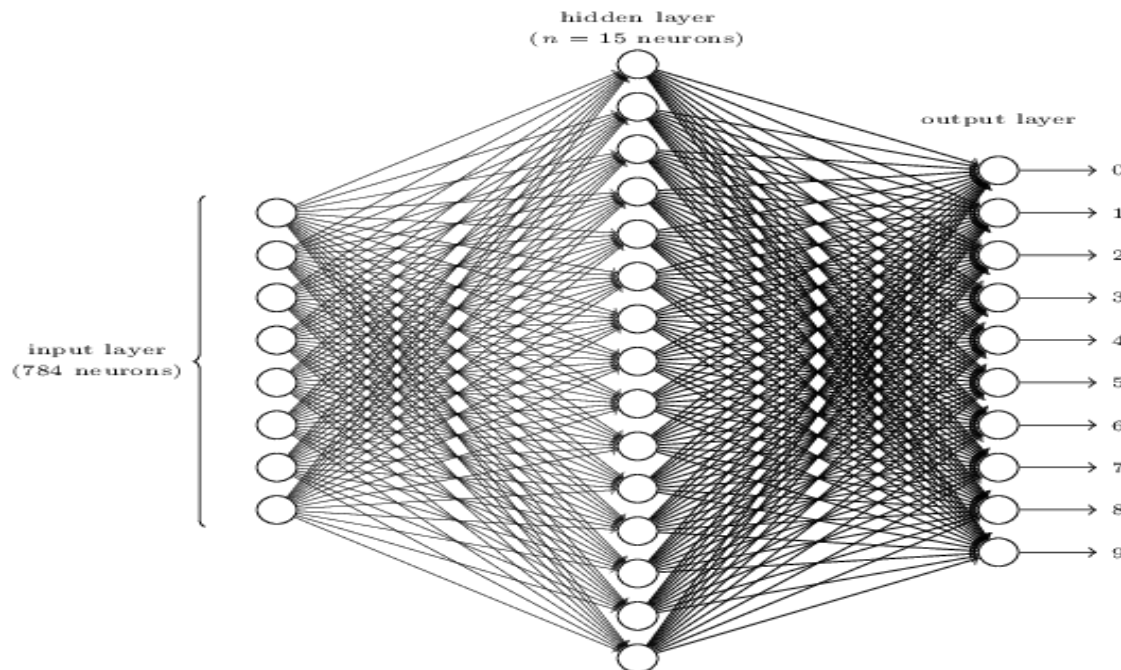


- Recognition (our focus)

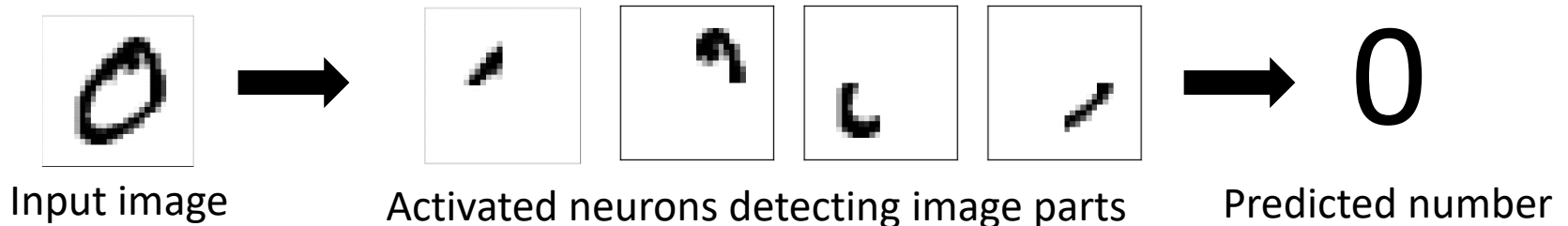


Digits Recognition Example


- The architecture of the used neural network



- What each neurons are doing?

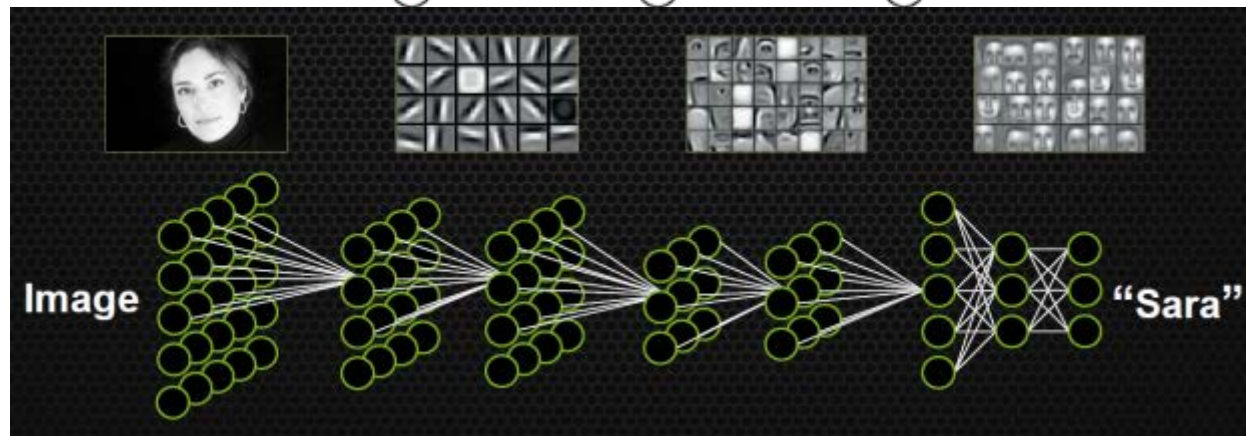
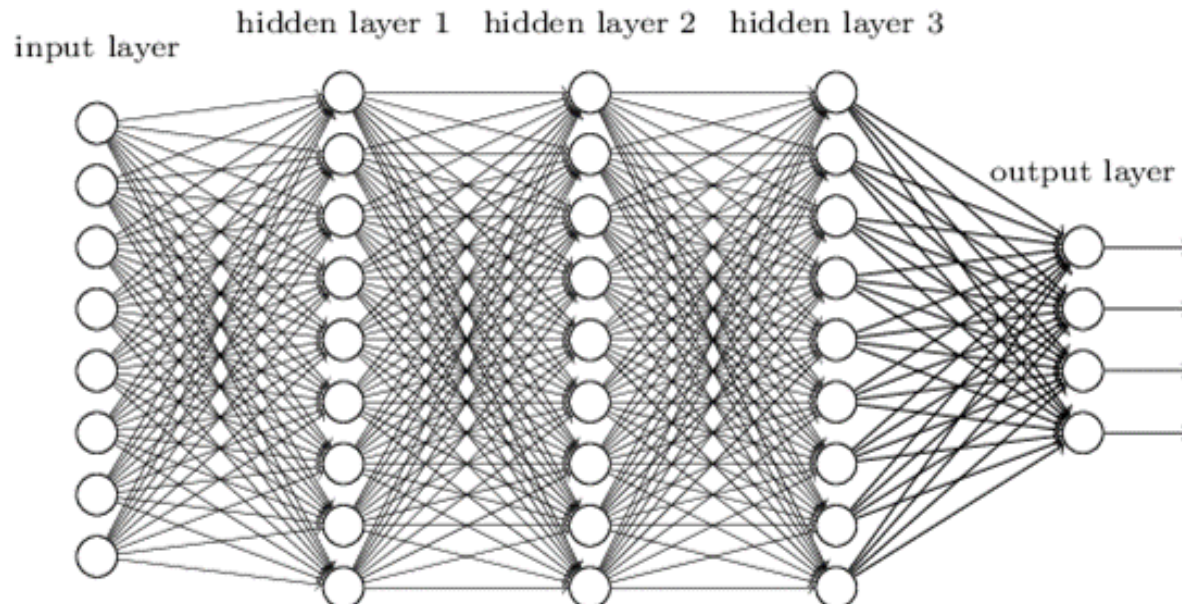


Neural Network

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

Towards Deep Learning

Deep neural network

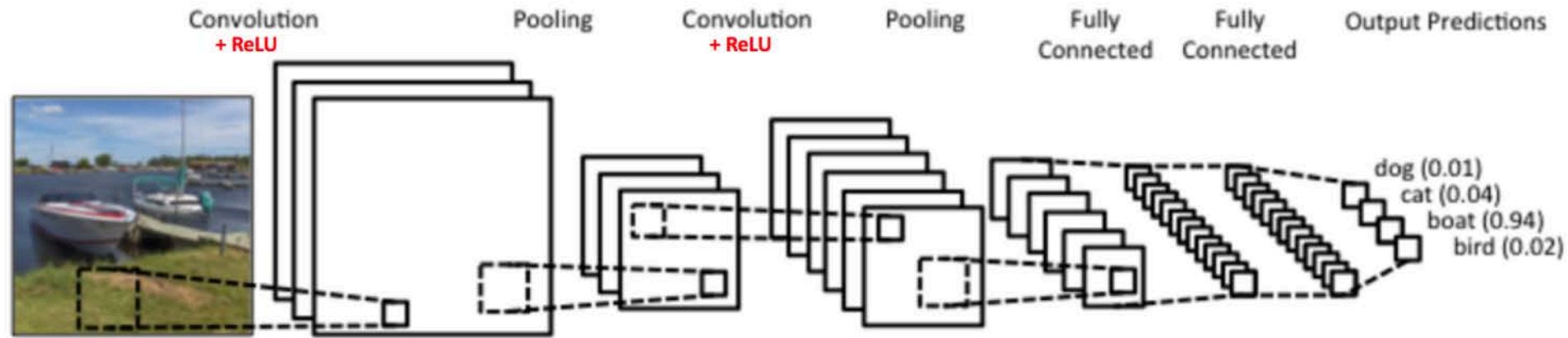


Popular Deep NNs

- 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

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Input

*

1	0	1
0	1	0
1	0	1

Filter

Convolution operator: weighted sum where weights are from filter matrix

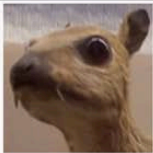



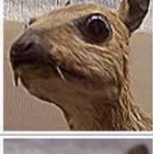
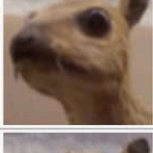
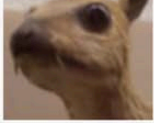
1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved
Feature

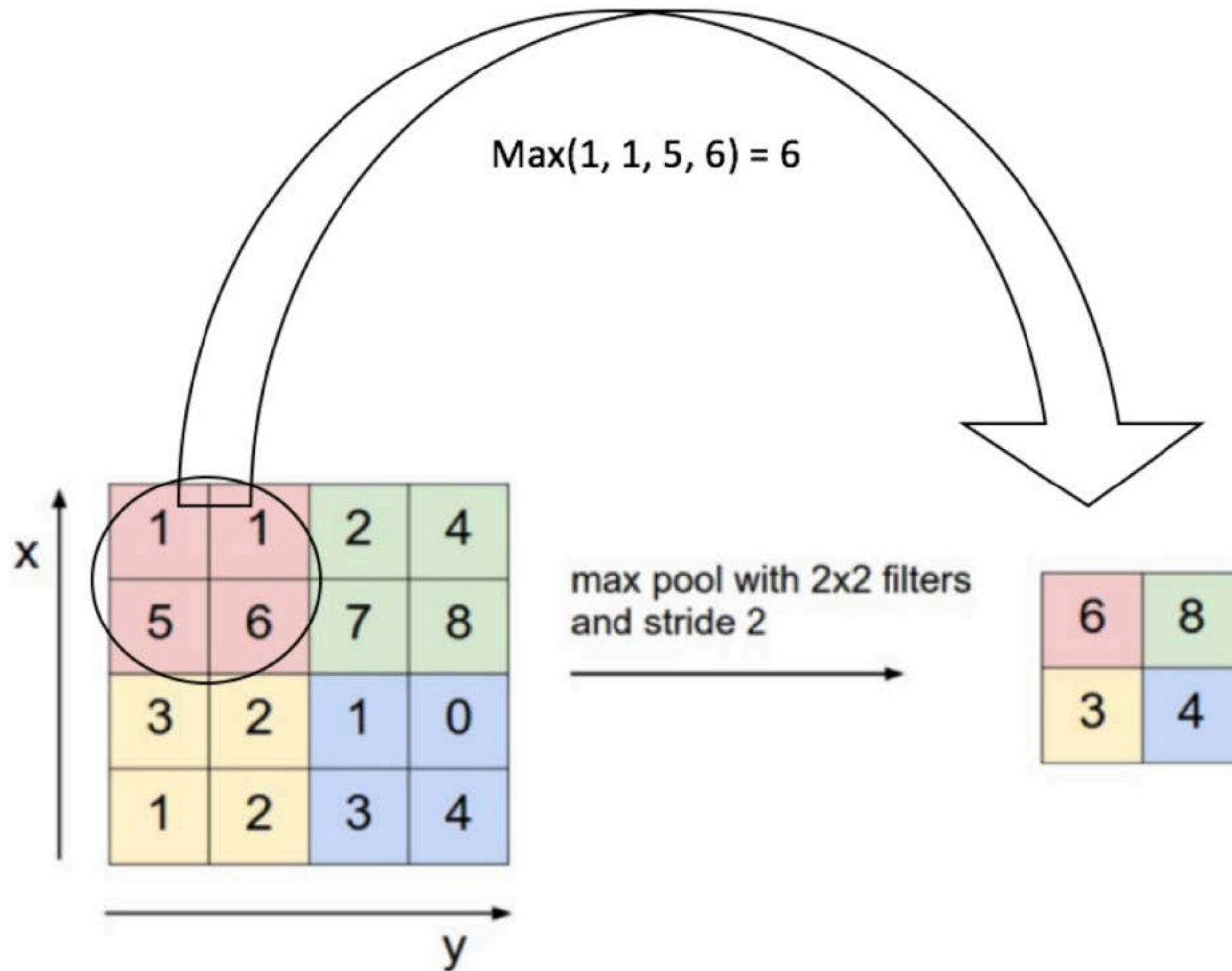
Effects of Different Filters

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_\(image_processing\)](https://en.wikipedia.org/wiki/Kernel_(image_processing))

Pooling

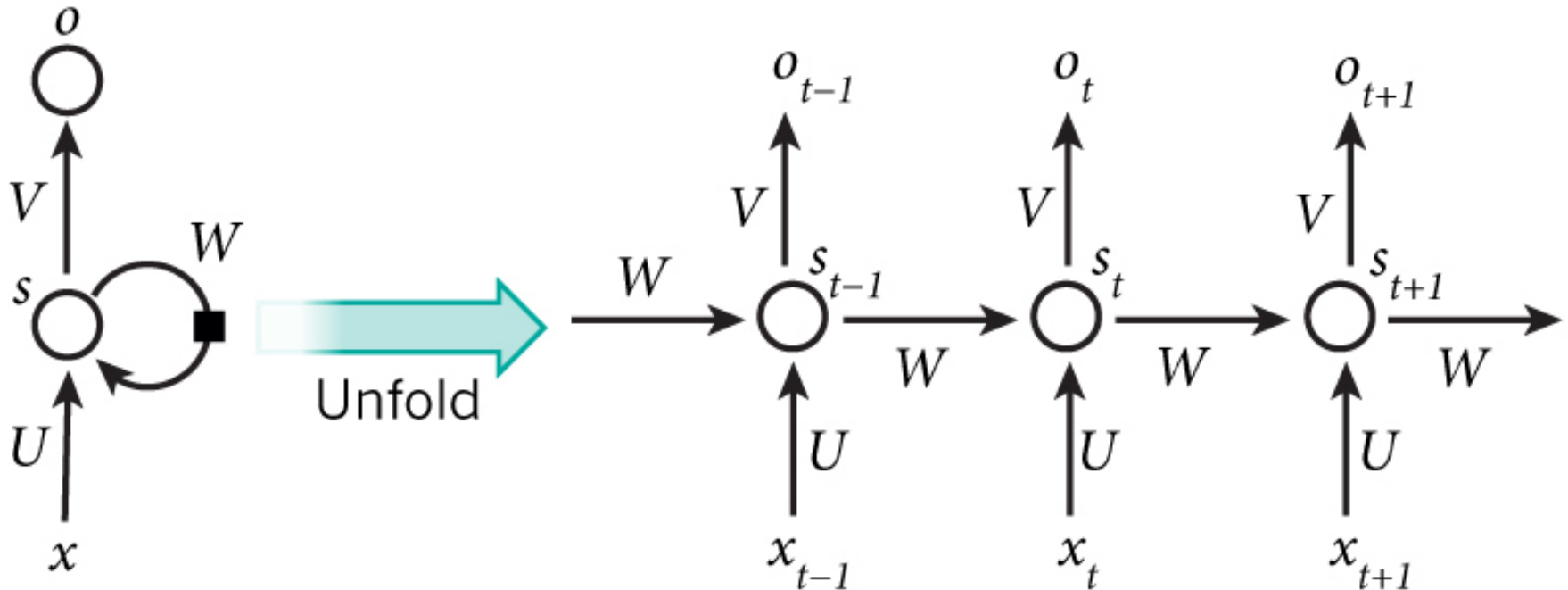


Rectified Feature Map

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

Recurrent Neural Network

- Model sequential information

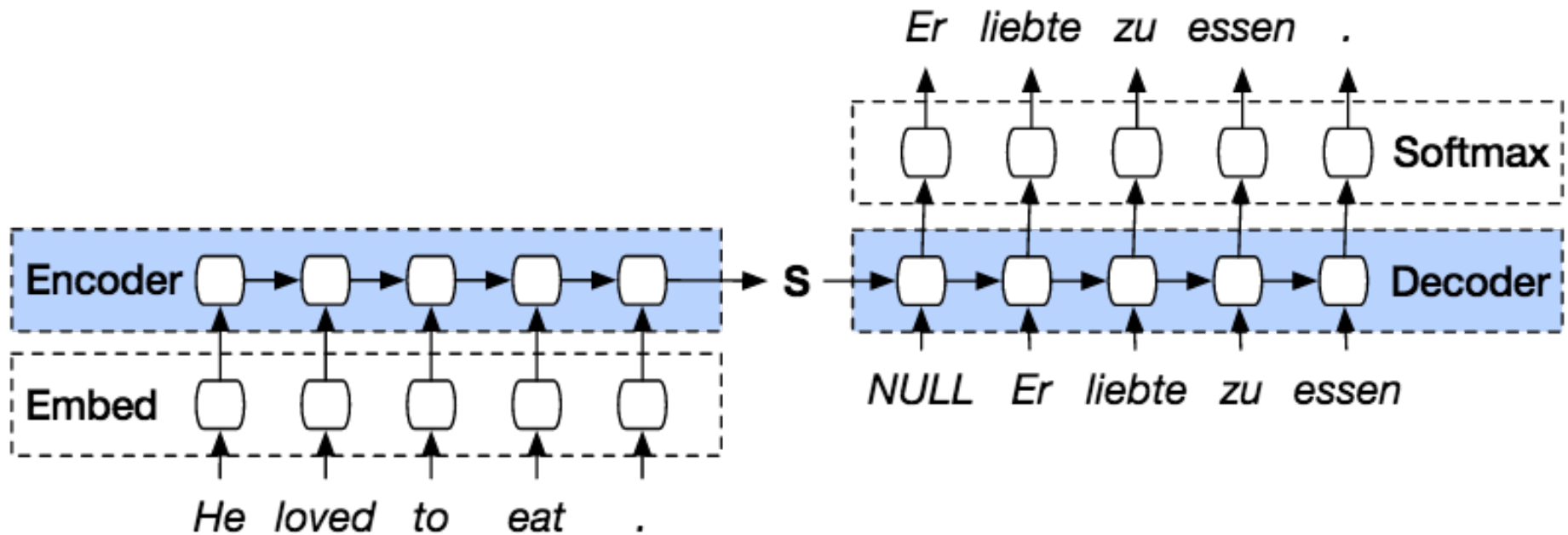


x_t : input vector at time t
 s_t : hidden state at time t
 o_t : output vector at time t


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

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_67000DxZCJB-3pi

- Deep Learning

- <http://neuralnetworksanddeeplearning.com/>

- <http://www.deeplearningbook.org/>

- <http://www.charuaggarwal.net/neural.htm>

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

Example (Cont.)

- Stochastic gradient for W^3 and b^3

- $$\frac{\partial E}{\partial W_{ij}^3} = \frac{\partial E}{\partial x_i^3} \frac{\partial x_i^3}{\partial z_i^3} \frac{\partial z_i^3}{\partial W_{ij}^3} = \underbrace{-(y_i - x_i^3) f'^3(z_i^3)}_{\delta_i^3} x_j^2$$

- $$\frac{\partial E}{\partial b_i^3} = \frac{\partial E}{\partial x_i^3} \frac{\partial x_i^3}{\partial z_i^3} \frac{\partial z_i^3}{\partial b_i^3} = \underbrace{-(y_i - x_i^3) f'^3(z_i^3)}_{\delta_i^3}$$

- Matrix form

- $$\frac{\partial E}{\partial W^3} = \frac{\partial E}{\partial x^3} \frac{\partial x^3}{\partial z^3} \frac{\partial z^3}{\partial W^3} = \underbrace{-(y - x^3) \circ f'^3(z^3)}_{\delta^3} (x^2)^T$$

◦: Hadamard product

Example (Cont.)

- Stochastic gradient for W^2

$$\begin{aligned} \bullet \frac{\partial E}{\partial W^2} &= \frac{\partial E}{\partial x^3} \frac{\partial x^3}{\partial z^3} \frac{\partial z^3}{\partial x^2} \frac{\partial x^2}{\partial z^2} \frac{\partial z^2}{\partial W^2} \\ &= -W^{3T} (y - x^3) \circ f'^3(z^3) \circ f'^2(z^2) x^{1T} \\ &= \underbrace{W^{3T} \delta^3 \circ f'^2(z^2)}_{\delta^2} x^{1T} \\ &= \delta^2 x^{1T} \end{aligned}$$

Example (Cont.)

- Stochastic gradient for W^1

- $$\frac{\partial E}{\partial W^1} = \underbrace{W^2{}^T \delta^2 \circ f'^1(z^1)}_{\delta^1} x^0{}^T$$