# CS247: ADVANCED DATA MINING

## 04: Neural Network and Deep Learning

---

**Instructor: Yizhou Sun**

yzsun@cs.ucla.edu

April 7, 2021

# Methods to Learn

| | Vector Data | Text Data | Graph & Network | Recommender Systems |
|---|---|---|---|---|
| **Classification** | **Naïve Bayes; Logistic Regression**; **NN** | | Label Propagation | |
| **Clustering** | **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 |

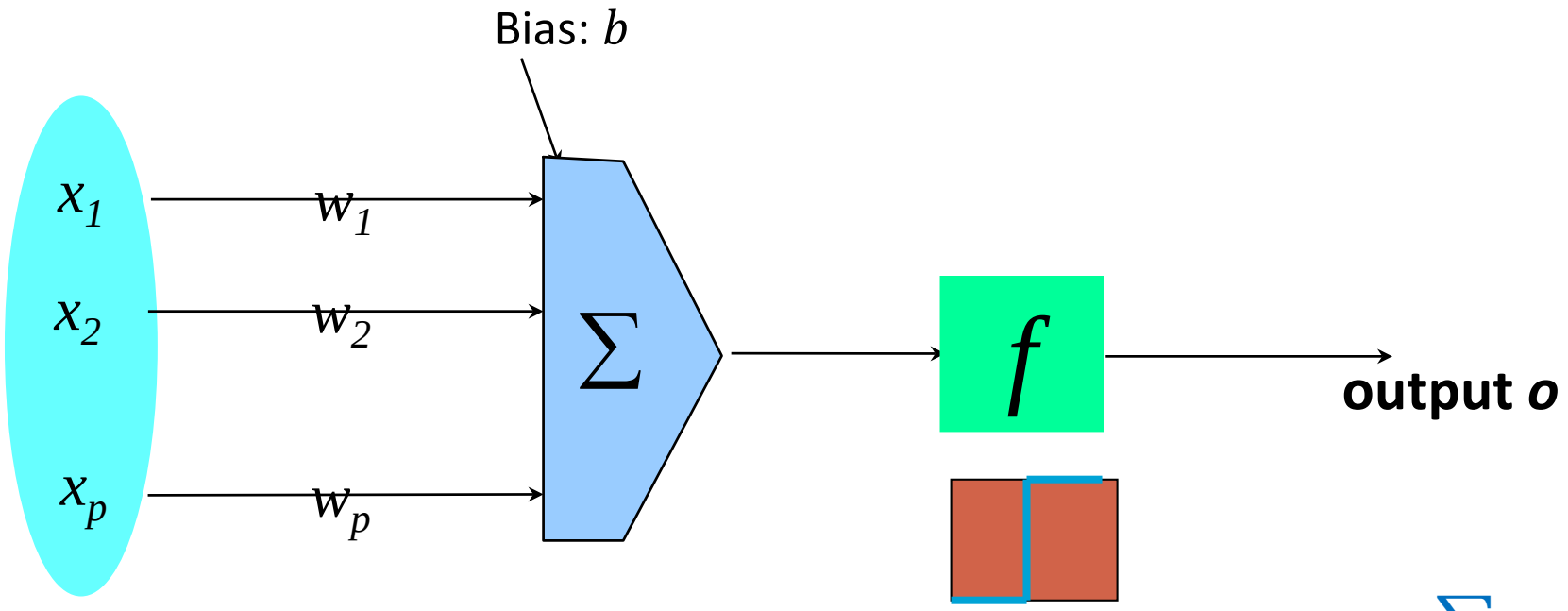
# Neural Network

- 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 ~.001 second
  - 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**

# Important Concepts

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

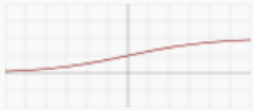
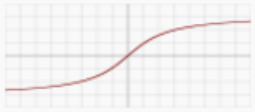
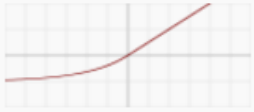
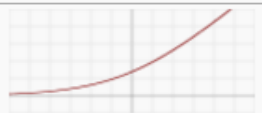
# Architecture

- 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
    - $\mathcal{L}(\boldsymbol{w}) = \frac{1}{n}\sum_i l\,(y_i, \hat{y}_i)$, 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
  - $l(y, \hat{y}) = \max(0, 1 - y\hat{y})$
  - $y \in \{-1,1\}, \hat{y} \in (-\infty, +\infty)$

# Optimization

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

# Regularization

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

# Neural Network

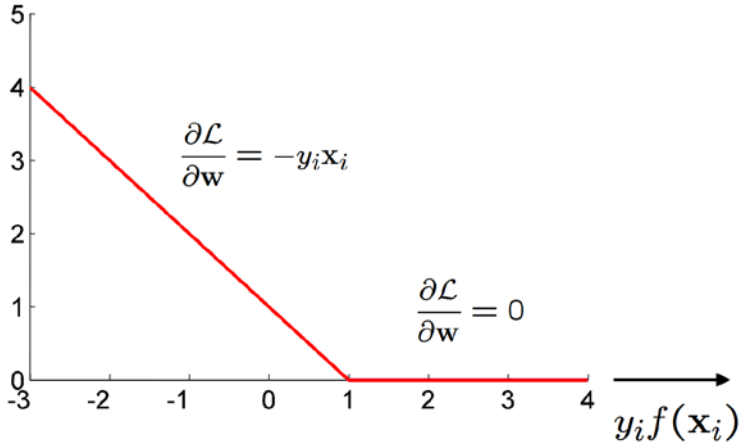
- Introduction
- Connection to Shallow Machine Learning Algorithms 
- Multi-Layer Feed-Forward Neural Network
- Deep Learning
- Summary

# 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
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta y_i \boldsymbol{x}_i$, for a misclassified training data point $(\boldsymbol{x}_i, y_i)$, i.e., $y_i w^T \boldsymbol{x}_i < 0$
  - $\eta$: learning rate

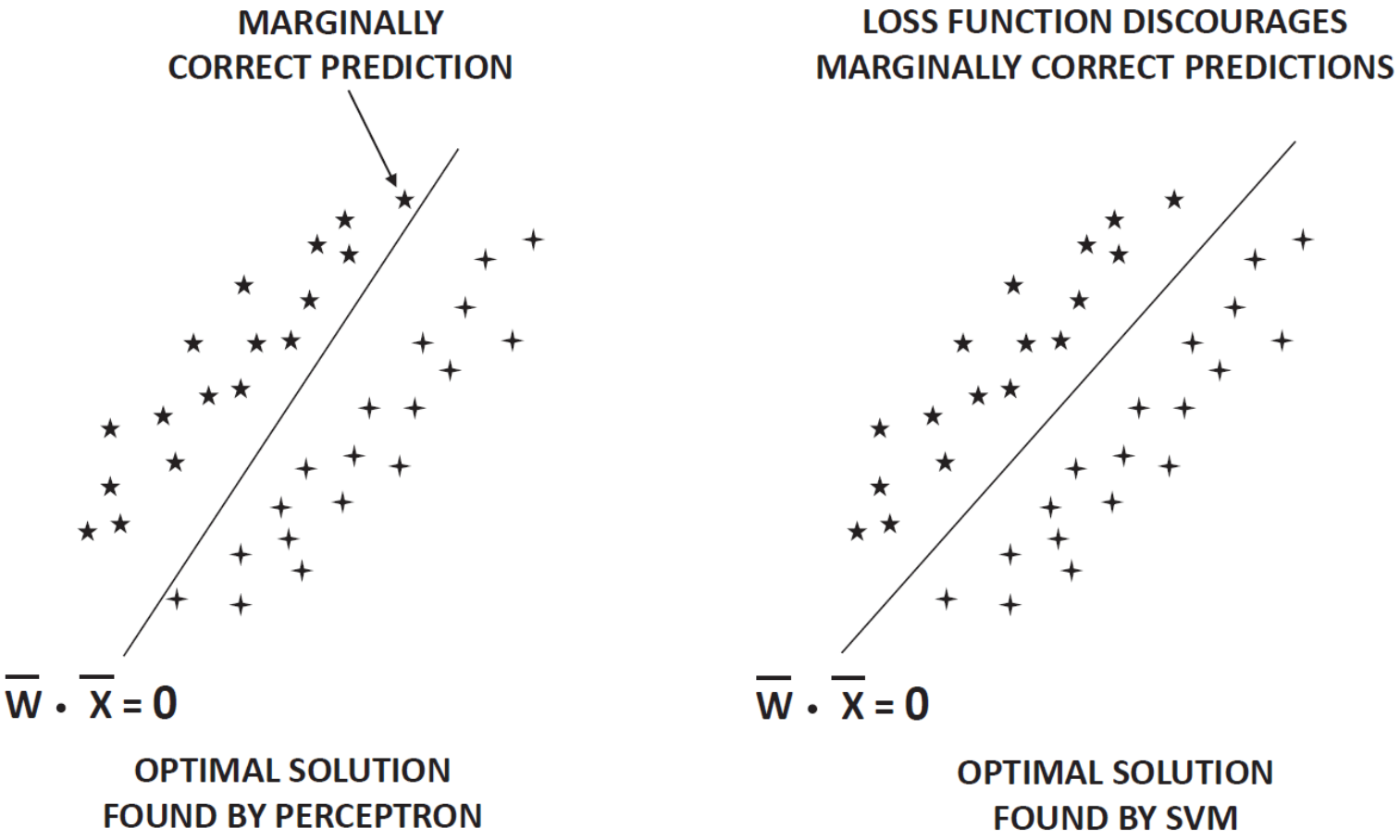
# Linear SVM

- 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., $\frac{1}{2}\lambda||\boldsymbol{w}||^2$
- Optimization
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta(\lambda w - y_i \boldsymbol{x}_i)$, 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
  - $\eta$: 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
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \left( y_i - \sigma\left( w^T \boldsymbol{x}_i \right) \right) \boldsymbol{x}_i$ , for a training data point $(\boldsymbol{x}_i, y_i)$
  - $\eta$: learning rate

# Question

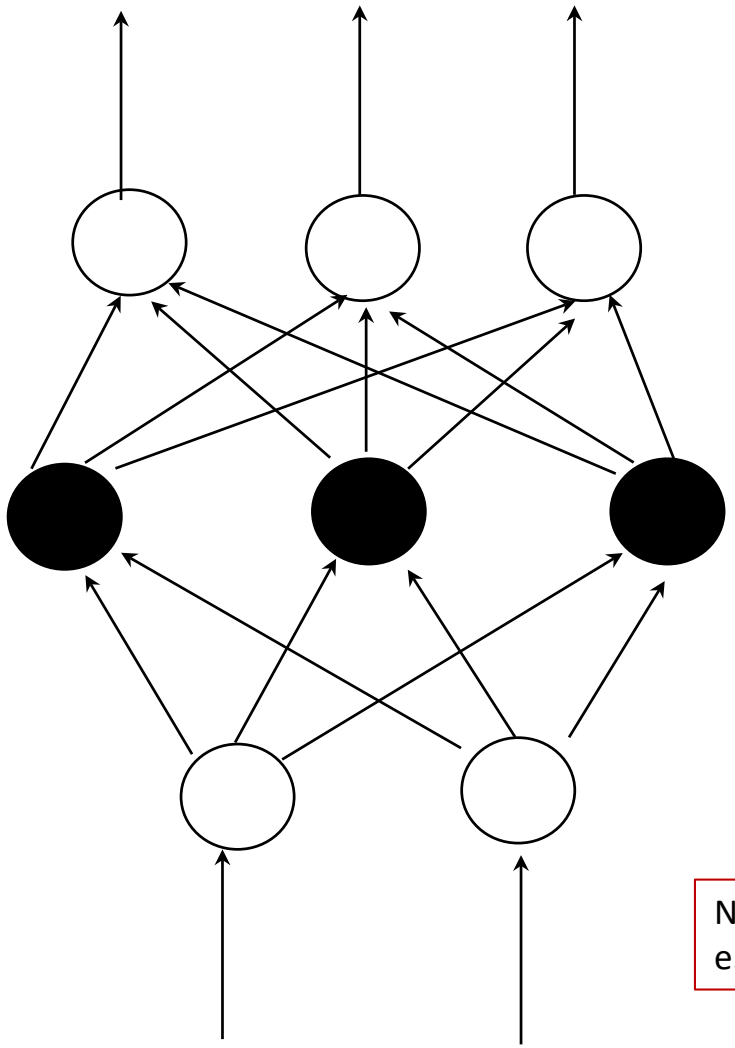
- How about multivariate logistic regression?

# 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: $d^{(1)} * d^{(0)}$

Nonlinear transformation,
e.g. sigmoid transformation

# How A Multi-Layer Neural Network Works

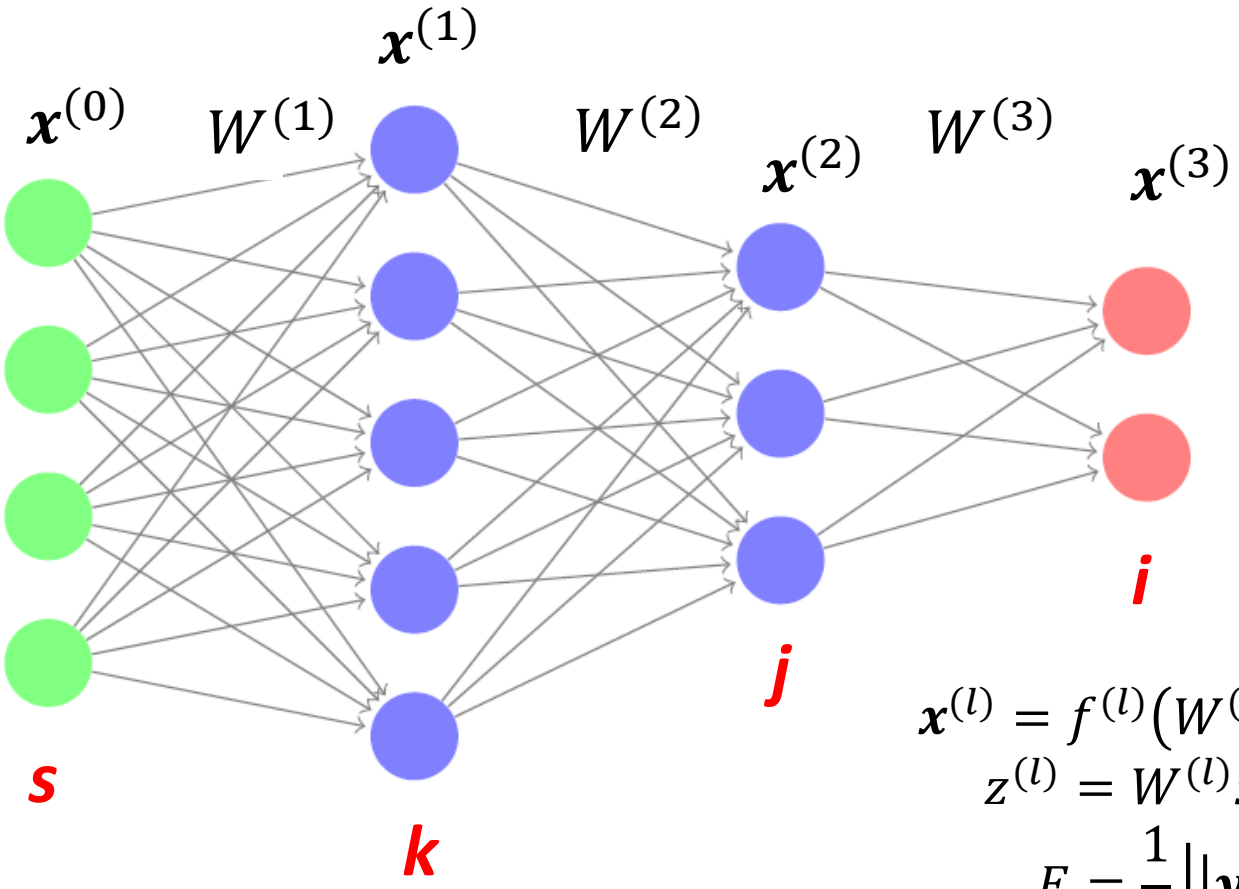
- The **inputs** to the network correspond to the attributes measured for each training tuple
- Inputs are fed simultaneously into the units making up the **input layer**
- They are then weighted and fed simultaneously to a **hidden layer**
  - The number of hidden layers is arbitrary
- The weighted 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: $E = \frac{1}{2}||\boldsymbol{y} - \hat{\boldsymbol{y}}||^2$




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

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

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

# Gradient for Layer 3 (Last Layer)

- Stochastic gradient for $W_{ij}^{(3)}$ and $b_i^{(3)}$
  - *Recall:*
    - $E = \frac{1}{2}\left\|\boldsymbol{y} - \boldsymbol{x}^{(3)}\right\|^2 = \frac{1}{2}\sum_i \left(y_i - x_i^{(3)}\right)^2$
    - $x_i^{(3)} = f^{(3)}(z_i^{(3)})$
    - $z_i^{(3)} = \sum_j W_{ij}^{(3)} x_j^{(2)} + b_i^{(3)}$
  - $\frac{\partial E}{\partial W_{ij}^{(3)}} = \frac{\partial E}{\partial x_i^{(3)}} \frac{\partial x_i^{(3)}}{\partial z_i^{(3)}} \frac{\partial z_i^{(3)}}{\partial W_{ij}^{(3)}} = \underbrace{-(y_i - x_i^{(3)})\mathrm{f}'^{(3)}\left(\mathrm{z}_\mathrm{i}^{(3)}\right)}_{\boldsymbol{\delta_i^{(3)}}} \mathrm{x}_j^{(2)}$
  - $\frac{\partial E}{\partial b_i^{(3)}} = \frac{\partial E}{\partial x_i^{(3)}} \frac{\partial x_i^{(3)}}{\partial z_i^{(3)}} \frac{\partial z_i^{(3)}}{\partial b_i^{(3)}} = -(y_i - x_i^{(3)})\mathrm{f}'^{(3)}\left(\mathrm{z}_\mathrm{i}^{(3)}\right)$

# Gradient for Layer 2

- Stochastic gradient for $W_{ij}^{(2)}$ and $b_i^{(2)}$
  - *Recall:*
    - $E = \frac{1}{2}\left|\left|\boldsymbol{y} - \boldsymbol{x}^{(3)}\right|\right|^2 = \frac{1}{2}\sum_i \left(y_i - x_i^{(3)}\right)^2$; $x_i^{(3)} = f^{(3)}\left(z_i^{(3)}\right)$; $z_i^{(3)} = \sum_j W_{ij}^{(3)} x_j^{(2)} + b_i^{(3)}$
    - $x_j^{(2)} = f^{(2)}\left(z_j^{(2)}\right)$; $z_j^{(2)} = \sum_k W_{jk}^{(2)} x_k^{(1)} + b_j^{(2)}$

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

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

# Gradient for Layer 1

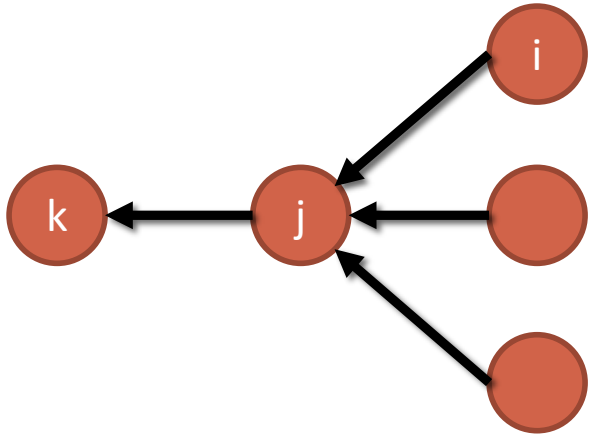
- Stochastic gradient for $W_{ks}^{(1)}$
  - $\frac{\partial E}{\partial W_{ks}^{(1)}} = \sum_j \delta_j^{(2)} W_{jk}^{(2)} f'^{(1)}\left(z_k^{(1)}\right) x_s^{(0)}$

# Question

- $\mathrm{k} \rightarrow j \rightarrow i$
- layer $l-1$ -> layer $l$ -> layer $l+1$
  - What would be a general formula for $W_{jk}^{(l)}$?

$$\delta_j^{(l)} = \sum_i \delta_i^{(l+1)} W_{ij}^{(l+1)} f'^{(l)}\left(z_j^{(l)}\right)$$

$$\frac{\partial E}{\partial W_{jk}^{(l)}} = \delta_j^{(l)} x_k^{(l-1)}$$

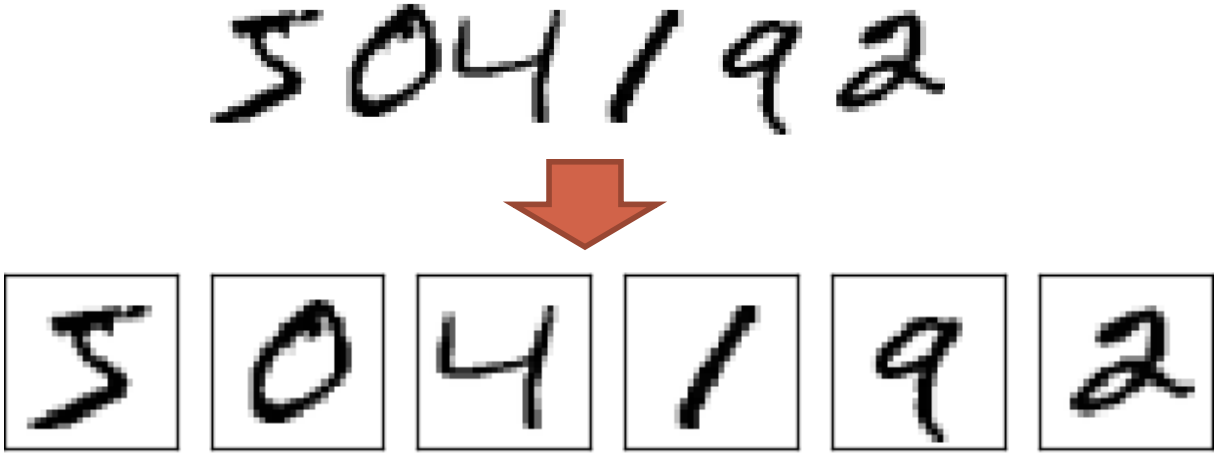


# Neural Network as a Classifier

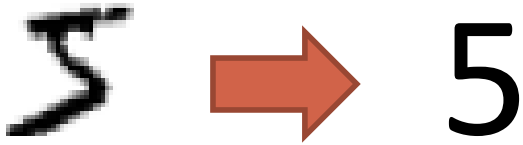
- Weakness
  - Long training time
  - Require a number of hyper-parameters typically best determined empirically, e.g., the network topology or "structure."
  - Poor interpretability: Difficult to interpret the symbolic meaning behind the learned weights and of "hidden units" in the network
- Strength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

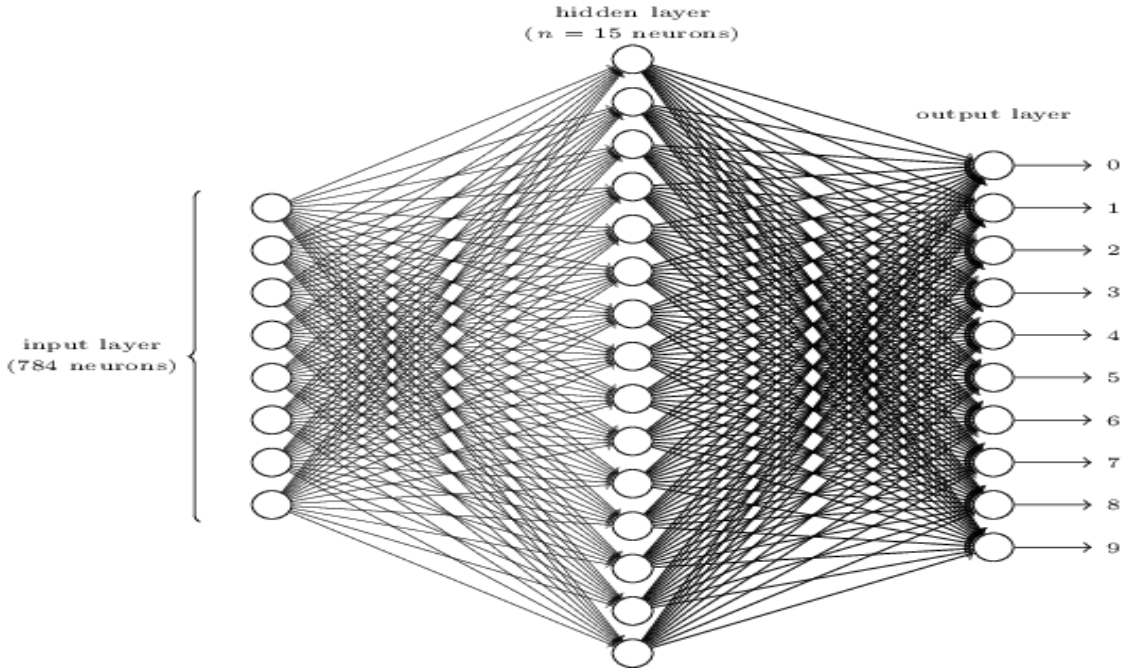


- Recognition (our focus)

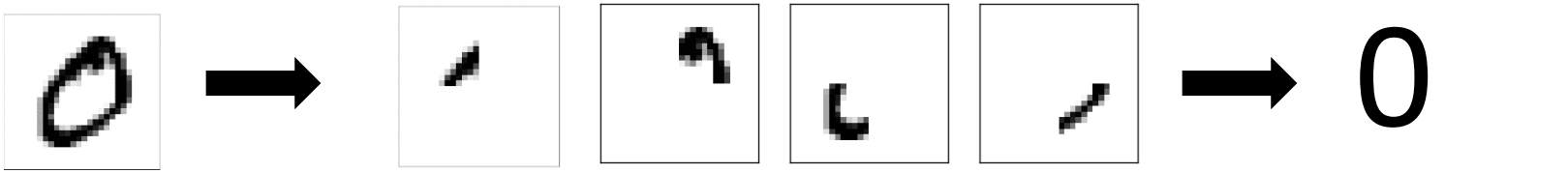


# Digits Recognition Example

- The architecture of the used neural network




- What each neurons are doing?



Input image

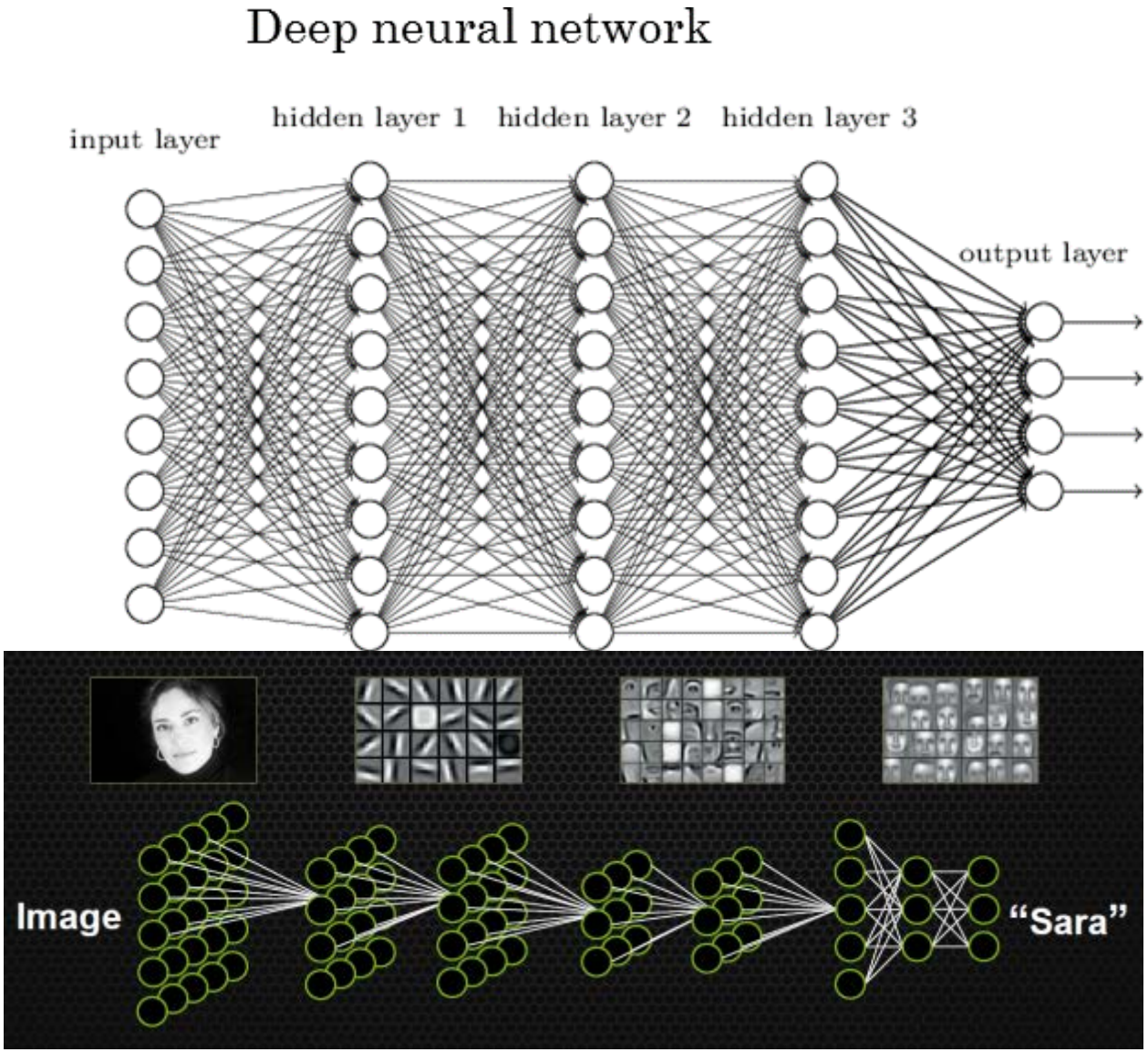
Activated neurons detecting image parts

Predicted number

# Neural Network

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

# Towards Deep Learning

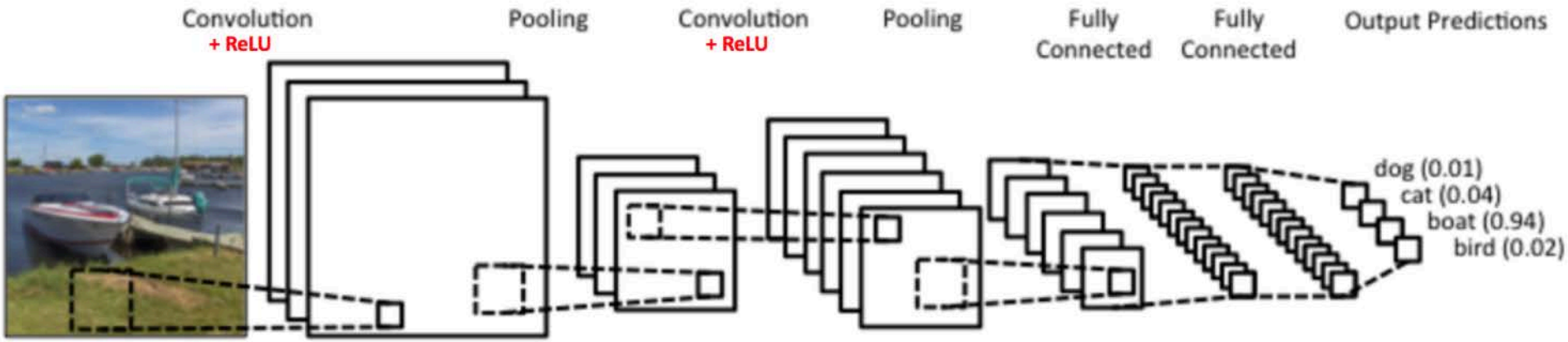


# Popular Deep NNs

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

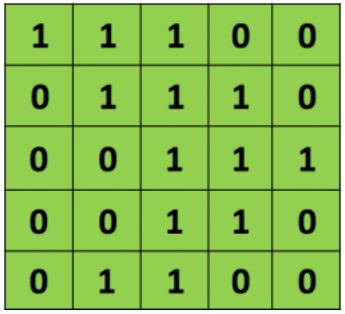
# Convolutional Neural Network

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



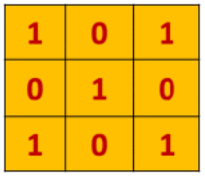

Source: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

# Convolution



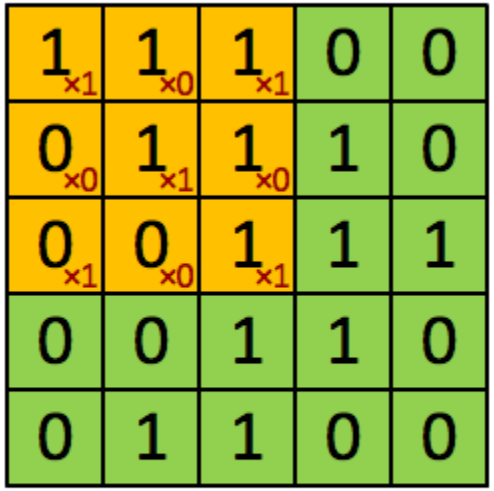

Input

*

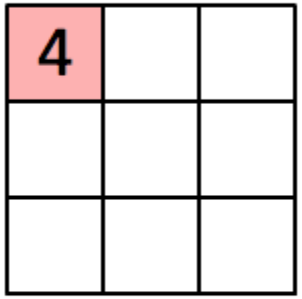



Filter

**Convolution operator: weighted sum where weights are from filter matrix**

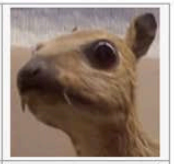
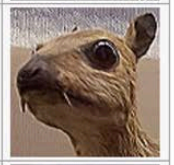
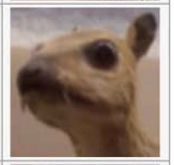
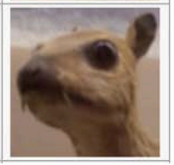



Image



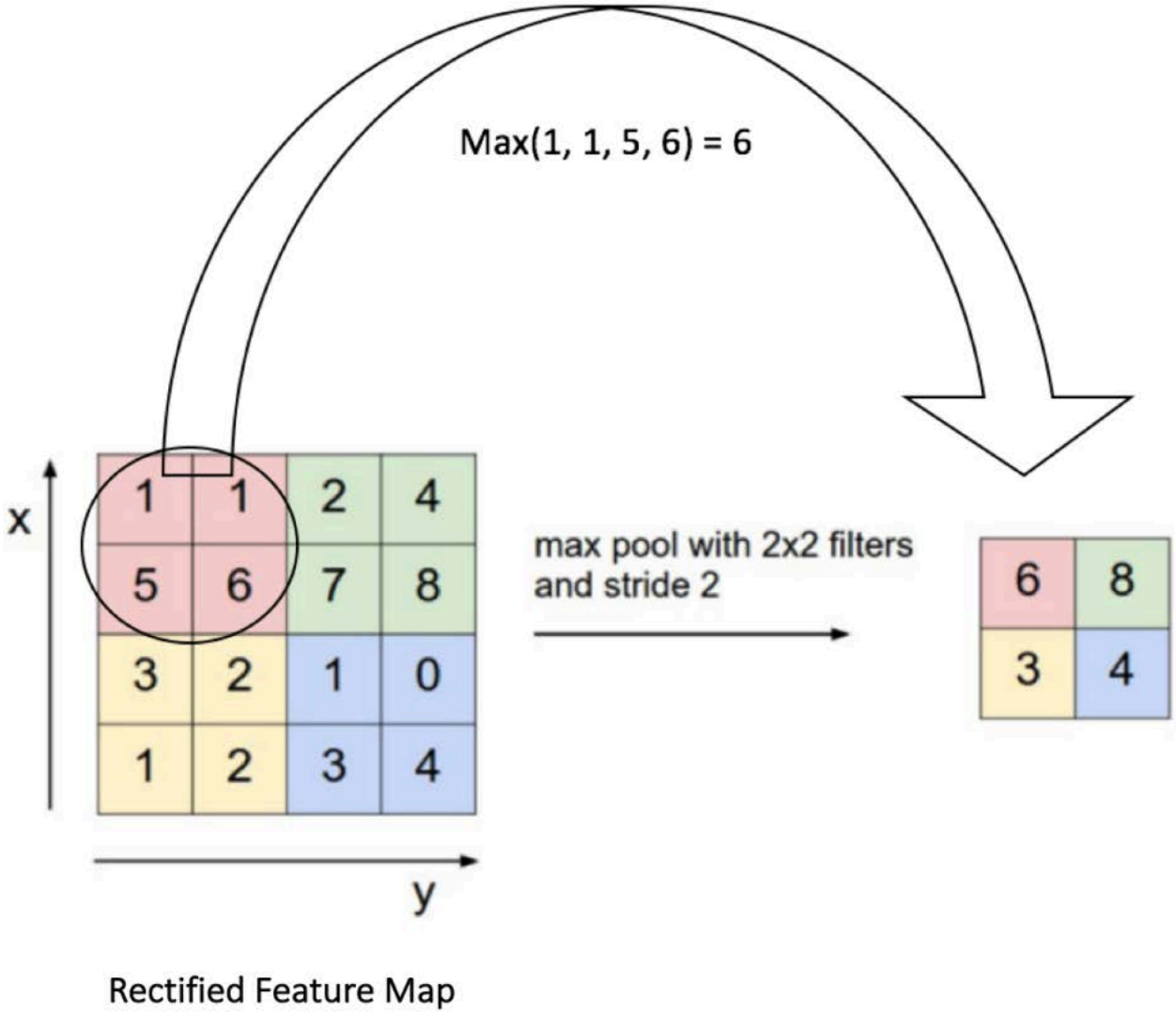

Convolved Feature

# 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)

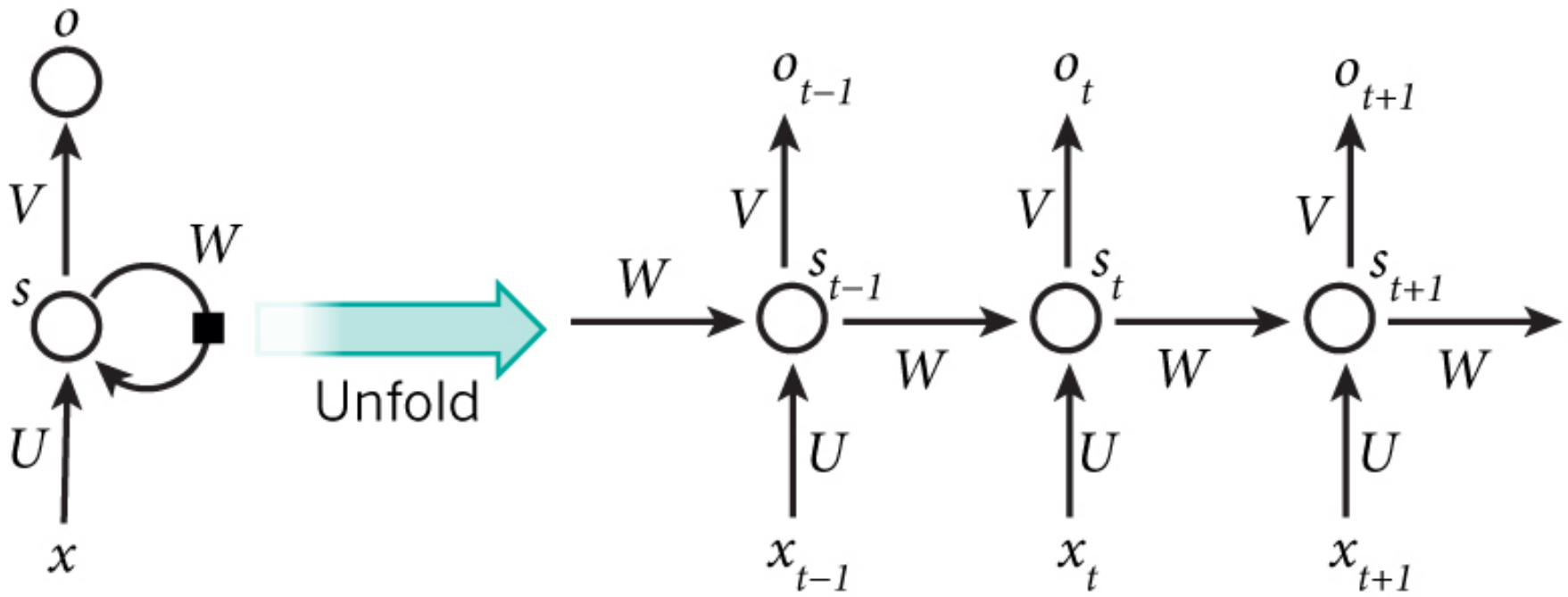
# Pooling




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

# Recurrent Neural Network

- Model sequential information

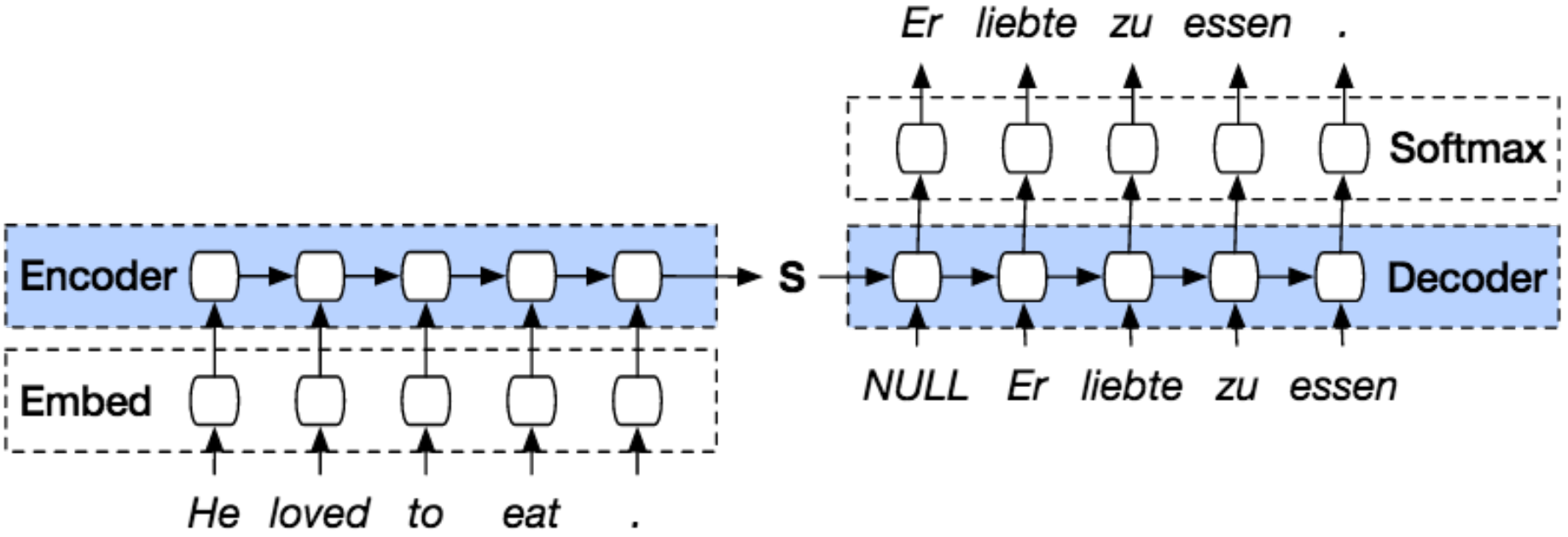



$x_t$: *input vector at time t*
$s_t$: *hidden state at time t*
$o_t$: *output vector at time t*

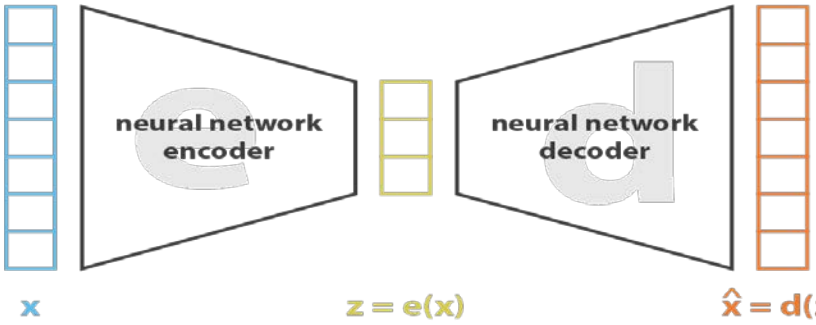
$$s_t = f(Ws_{t-1} + Ux_t)$$
$$o_t = g(Vs_t)$$

# Application of Machine Translation

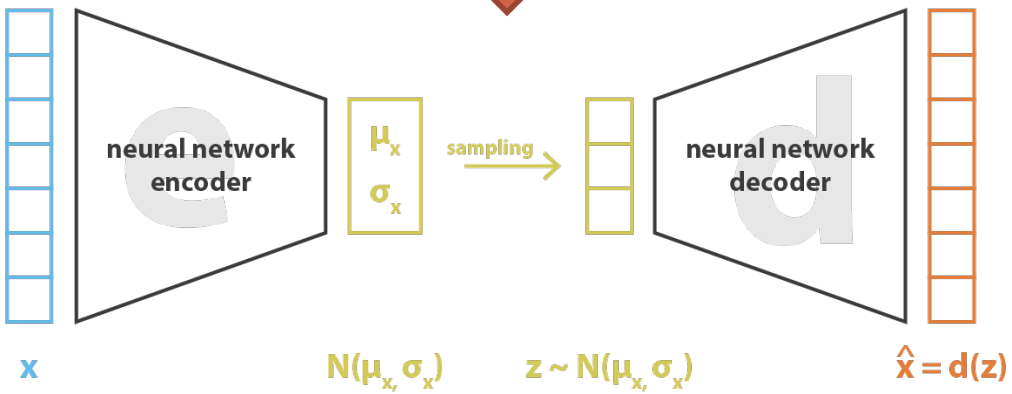
- Seq2Seq model



# Variational Autoencoder




$$\text{loss} = \| x - \hat{x} \|^2 = \| x - d(z) \|^2 = \| x - d(e(x)) \|^2$$




$$\text{loss} = \| x - \hat{x} \|^2 + KL[\, N(\mu_x, \sigma_x), N(0, I) \,] = \| x - d(z) \|^2 + KL[\, N(\mu_x, \sigma_x), N(0, I) \,]$$

https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

# 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

# Example (Cont.)

- Stochastic gradient for $W^3$ and $b^3$
  - $\frac{\partial E}{\partial W_{ij}^3} = \frac{\partial E}{\partial x_i^3}\frac{\partial x_i^3}{\partial z_i^3}\frac{\partial z_i^3}{\partial W_{ij}^3} = \underbrace{-(y_i - x_i^3)\mathrm{f}'^3(\mathrm{z}_\mathrm{i}^3)}_{\delta_i^3}\mathrm{x}_j^2$
  - $\frac{\partial E}{\partial b_i^3} = \frac{\partial E}{\partial x_i^3}\frac{\partial x_i^3}{\partial z_i^3}\frac{\partial z_i^3}{\partial b_i^3} = -(y_i - x_i^3)\mathrm{f}'^3(\mathrm{z}_\mathrm{i}^3)$
  - Matrix form
    - $\circ$: Hadamard product
    - $\frac{\partial E}{\partial W^3} = \frac{\partial E}{\partial x^3}\frac{\partial x^3}{\partial z^3}\frac{\partial z^3}{\partial W^3} = \underbrace{-(y - x^3) \circ \mathrm{f}'^3(\mathrm{z}^3)}_{\delta^3}(\mathrm{x}^2)^\mathrm{T}$

# Example (Cont.)

- Stochastic gradient for $W^2$
  - $$\frac{\partial E}{\partial W^2} = \frac{\partial E}{\partial x^3}\frac{\partial x^3}{\partial z^3}\frac{\partial z^3}{\partial x^2}\frac{\partial x^2}{\partial z^2}\frac{\partial z^2}{\partial W^2}$$

$$= -{W^3}^T\left(y - x^3\right) \circ \mathrm{f}'^3\left(\mathrm{z}^3\right) \circ \mathrm{f}'^2\left(\mathrm{z}^2\right){x^1}^T$$

$$= \underbrace{{W^3}^T\delta^3 \circ \mathrm{f}'^2\left(\mathrm{z}^2\right)}_{\delta^2}{x^1}^T$$

$$= \delta^2{x^1}^T$$

# Example (Cont.)

- Stochastic gradient for $W^1$
  - $\frac{\partial E}{\partial W^1} = \underbrace{W^{2^T} \delta^2 \circ \mathrm{f}'^1\left(\mathrm{z}^1\right)}_{\boldsymbol{\delta^1}} x^{0^T}$