

CS247: ADVANCED DATA MINING


04: Basics: Neural Network and Deep Learning

Instructor: Yizhou Sun

yzsun@cs.ucla.edu

January 18, 2022

Neural Network

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

A little bit of History

Beginnings

Thresholded Logic Unit

1943

Perceptron

1957

Adaline

1960

1940

1950

1960



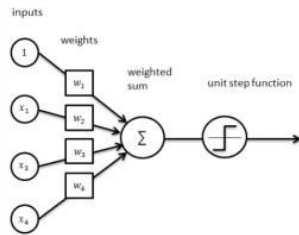
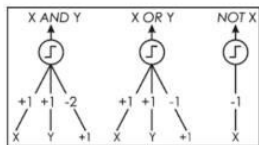
S. McCulloch - W. Pitts



R. Rosenblatt



B. Widrow - M. Hoff



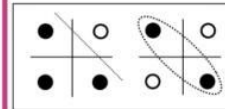
1st Neural Winter

XOR Problem

1969



M. Minsky - S. Papert



2nd Neural Winter

Multilayer Backprop

1982

CNNs

1986

LSTMs

1989

SVMs

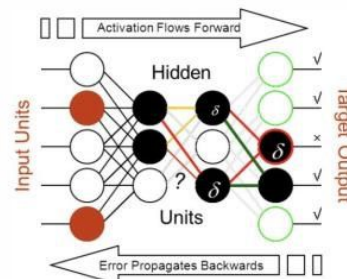
1997



P. Werbos

D. Rumelhart - G. Hinton - R. Williams

Y. Lecun - J. Schmidhuber



GPU Era

Deep Nets

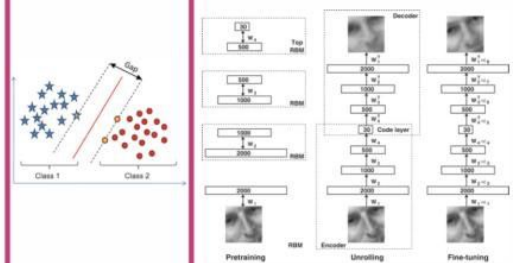
2006

Alex Net

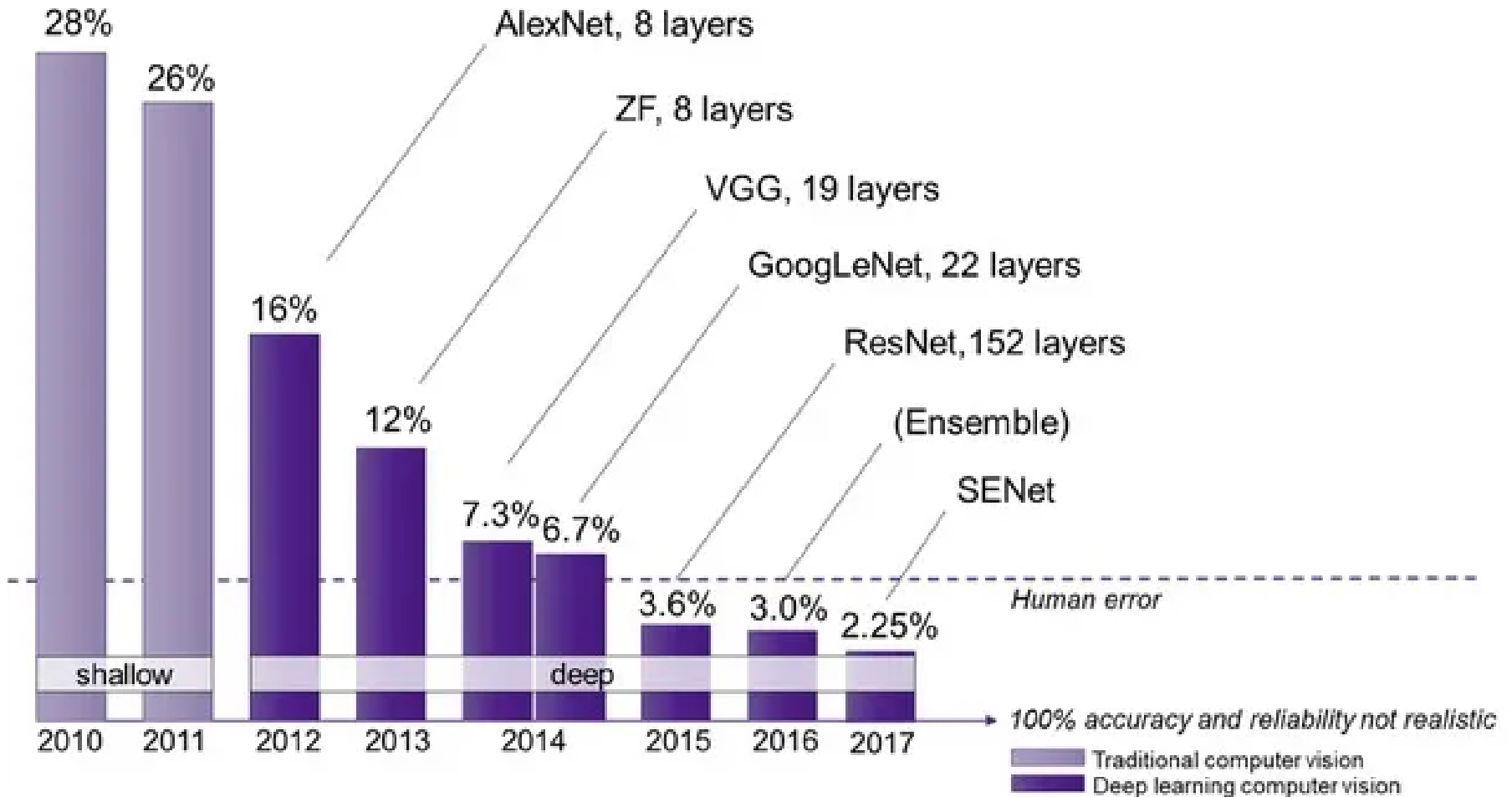
2012



R. Salakhutdinov - J. Hinton - A. Krizhevsky - I. Sutskever



ImageNet Challenge



<https://semiengineering.com/new-vision-technologies-for-real-world-applications/>

AlphaGo

Master of Go Board Game Is Walloped by Google Computer Program



By [Choe Sang-Hun](#) and [John Markoff](#)

March 9, 2016



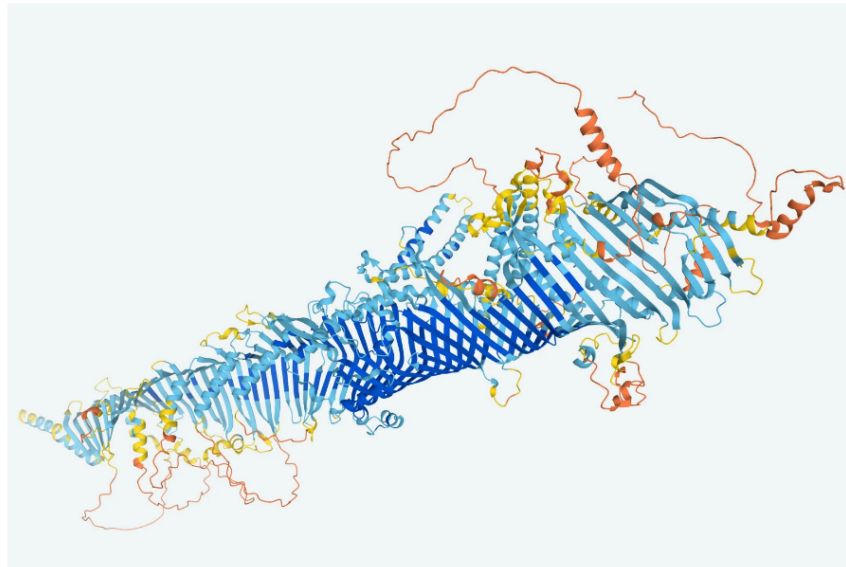
SEOUL, South Korea — Computer, one. Human, zero.

A [Google](#) computer program stunned one of the world's top players on Wednesday in a round of Go, which is believed to be the most complex board game ever created.

AlphaFold

A.I. Predicts the Shapes of Molecules to Come

DeepMind has given 3-D structure to 350,000 proteins, including every one made by humans, promising a boon for medicine and drug design.



Proteins belonging to a fruit fly are given shape by DeepMind's A.I. technology. The company's new database includes the 3-D structures for all proteins expressed by the human genome as well as those appearing in 20 other organisms. DeepMind

GPT-3



The New York Times

Account ▾

Meet GPT-3. It Has Learned to Code (and Blog and Argue).

The latest natural-language system generates tweets, pens poetry, summarizes emails, answers trivia questions, translates languages and even writes its own computer programs.

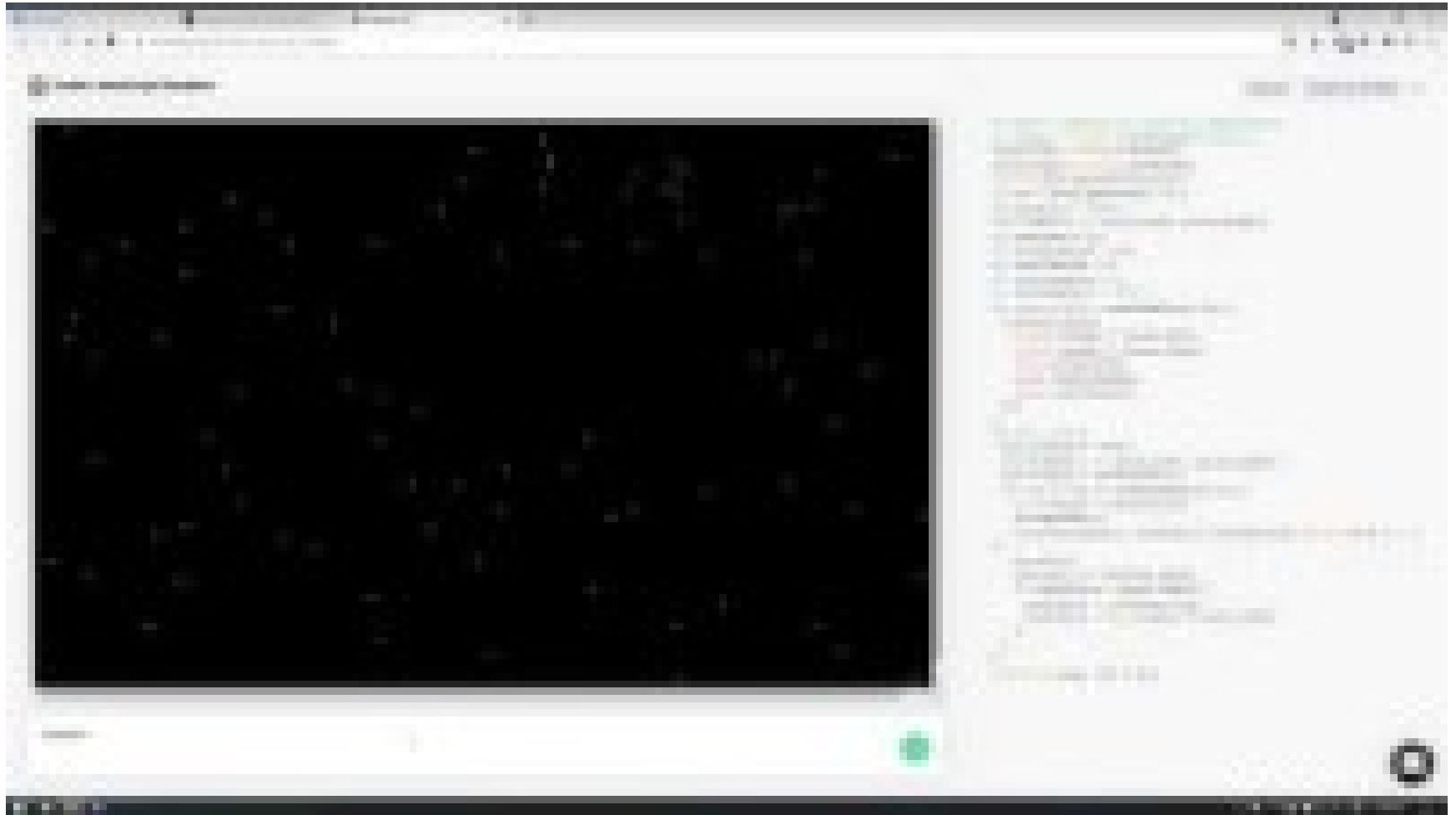


By **Cade Metz**

Nov. 24, 2020

This summer, an artificial intelligence lab in San Francisco called OpenAI unveiled a technology several months in the making. This new system, GPT-3, had spent the previous months learning the ins and

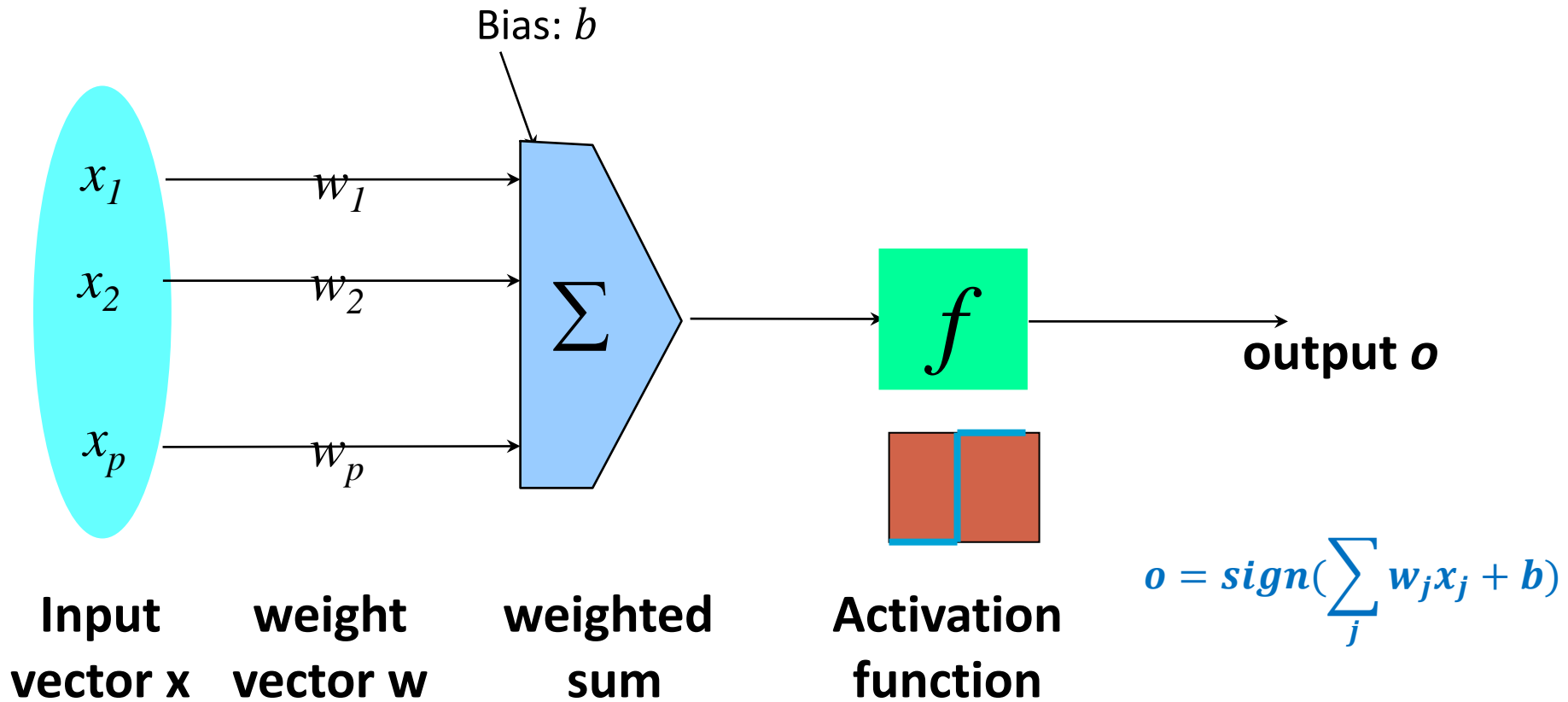
OpenAI: Codex



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


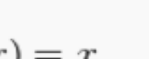



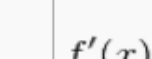



Architecture

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

slido



Audience Q&A Session

ⓘ Start presenting to display the audience questions on this slide.

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

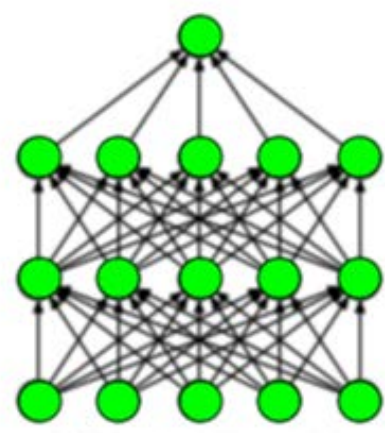
- Squared error
 - $l(y, \hat{y}) = (y - \hat{y})^2$
- (Binary) cross entropy loss
 - $l(y, \hat{y}) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$
 - $y \in \{0, 1\}, \hat{y} \in [0, 1]$
- Hinge loss
 - $l(y, \hat{y}) = \max(0, 1 - y\hat{y})$
 - $y \in \{-1, 1\}, \hat{y} \in (-\infty, +\infty)$

Optimization

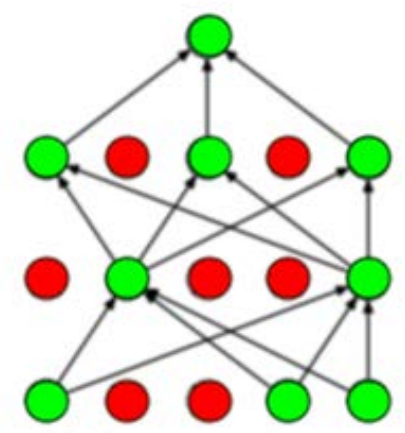
- Given a training dataset, minimize the empirical risk
 - Find \mathbf{w} , such that $\mathcal{L}(\mathbf{w}) = \frac{1}{n} \sum_i l(y^{(i)}, \hat{y}^{(i)})$ is minimized
- Solution:
 - Stochastic gradient descent + chain rule = backpropagation
 - $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla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

Regularization

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




(a) Standard Neural Net



(b) After applying dropout.

<https://medium.com/analytics-vidhya/a-simple-introduction-to-dropout-regularization-with-code-5279489dda1e>

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., $y^{(i)} \mathbf{w}^T \mathbf{x}^{(i)} \leq 0$
 - η : learning rate

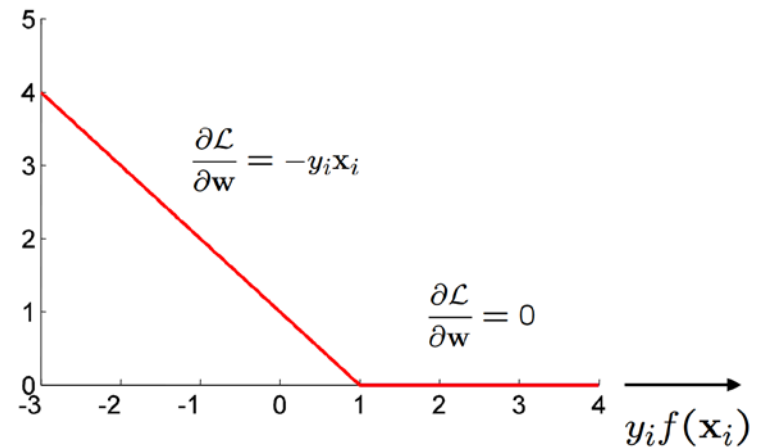
Example: 1 for “Y” and -1 for “N”;

$$\eta = 0.9$$

x0	x1	x2	true label	w before update	predicted label	w after update
1	0	1	Y	(0.0, 0.0, 0.0)	N	(0.9, 0.0, 0.9)
1	1	1	N	(0.9, 0.0, 0.9)	Y	(0.0, -0.9, 0.0)
1	0	0	Y	(0.0, -0.9, 0.0)	N	(0.9, -0.9, 0.0)
1	1	0	Y	(0.9, -0.9, 0.0)	N	(1.8, 0.0, 0.0)
1	0	1	Y	(1.8, 0.0, 0.0)	Y	(1.8, 0.0, 0.0)
1	1	1	N	(1.8, 0.0, 0.0)	Y	(0.9, -0.9, -0.9)
1	0	0	Y	(0.9, -0.9, -0.9)	Y	(0.9, -0.9, -0.9)
1	1	0	Y	(0.9, -0.9, -0.9)	N	(1.8, 0.0, -0.9)
1	0	1	Y	(1.8, 0.0, -0.9)	Y	(1.8, 0.0, -0.9)
1	1	1	N	(1.8, 0.0, -0.9)	Y	(0.9, -0.9, -1.8)
1	0	0	Y	(0.9, -0.9, -1.8)	Y	(0.9, -0.9, -1.8)
1	1	0	Y	(0.9, -0.9, -1.8)	N	(1.8, 0.0, -1.8)

Linear SVM

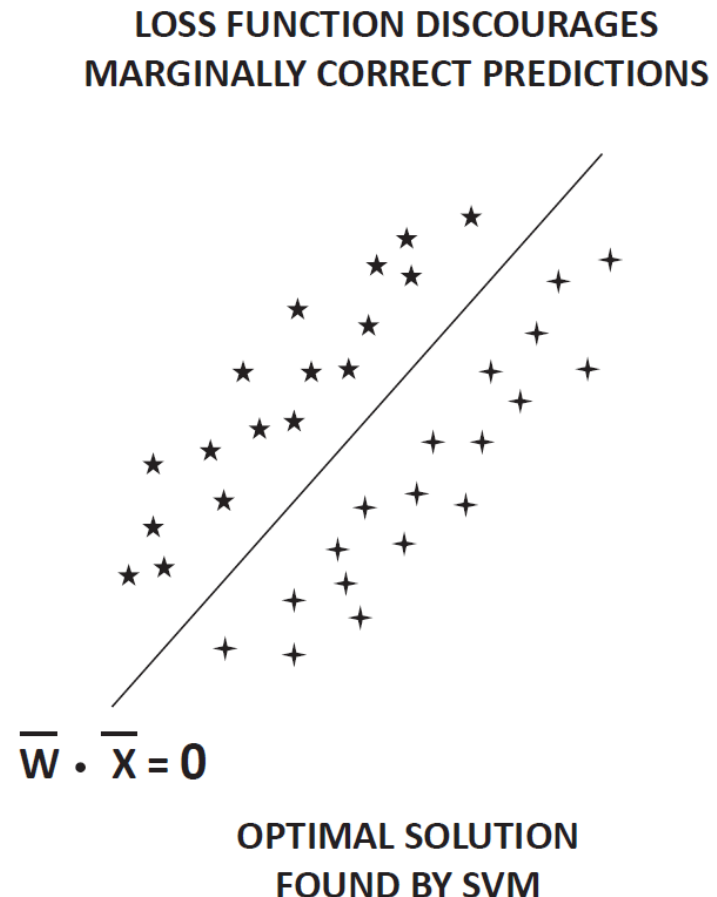
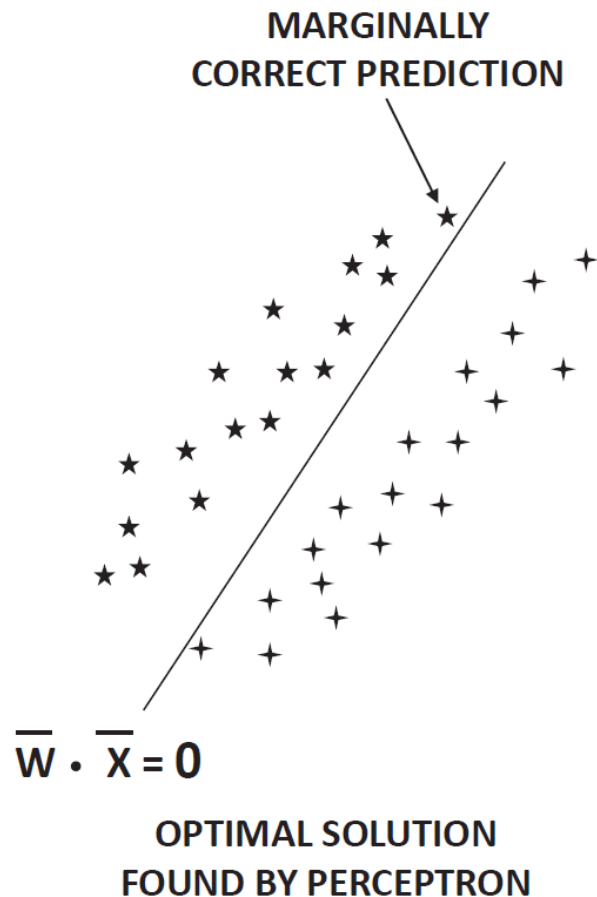
- Architecture:
 - A single neuron
- Activation function
 - Training: identity function
 - Inference: sign function/step function
- Loss function
 - $l(y, \hat{y}) = \max(0, 1 - y\hat{y})$
- Regularization
 - L2, i.e., $\frac{1}{2} \lambda \|\mathbf{w}\|^2$
- Optimization
 - $\mathbf{w} \leftarrow \mathbf{w} - \eta(\lambda \mathbf{w} - y_i \mathbf{x}_i)$, 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>



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

slido



For logistic regression, is minimizing the cross-entropy loss function equivalent to maximizing the log-likelihood function introduced in Lecture 2?

ⓘ Start presenting to display the poll results on this slide.

Question

- How about multivariate logistic regression?


slido



Audience Q&A Session

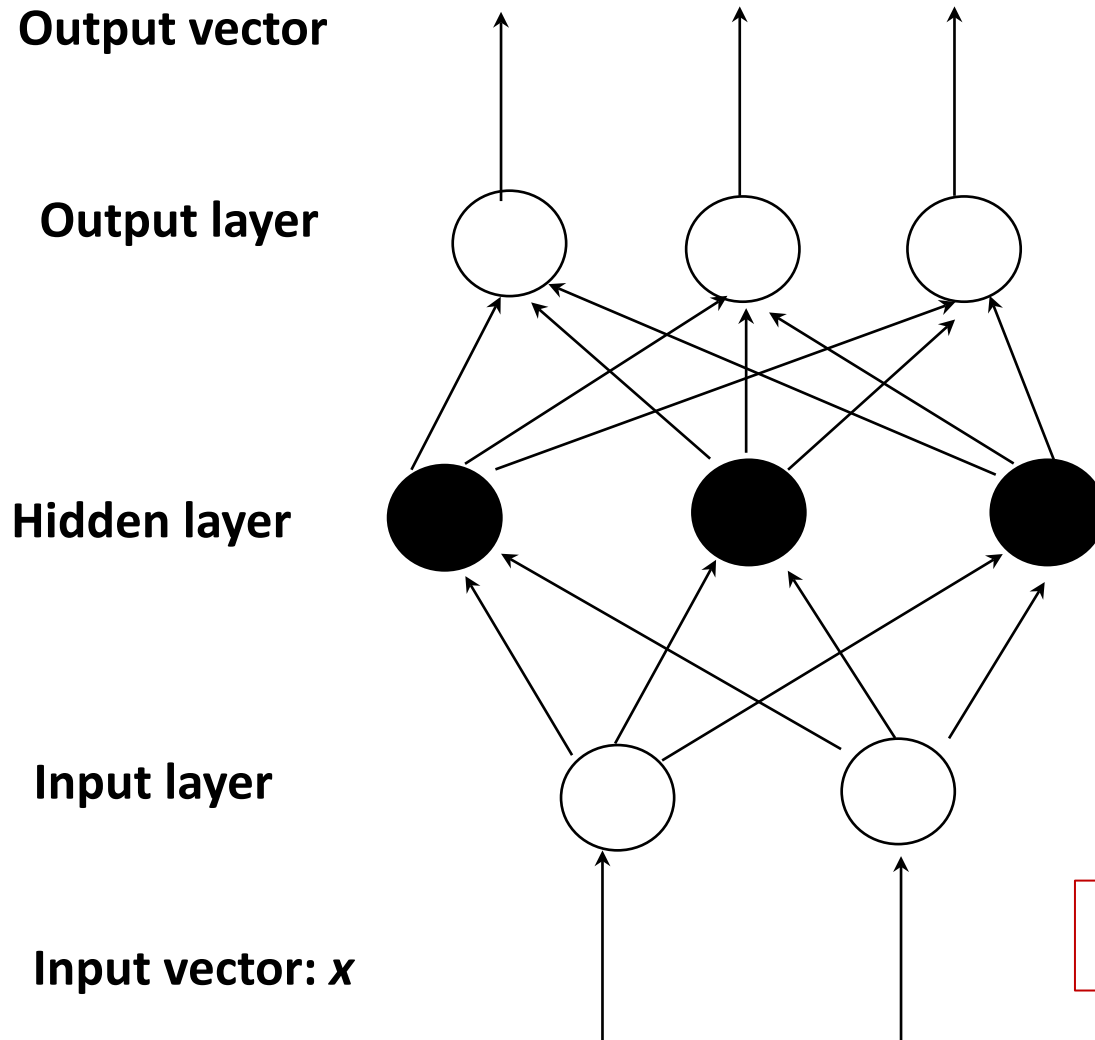
① Start presenting to display the audience questions on this slide.

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

Nonlinear transformation,
e.g. sigmoid transformation

How A Multi-Layer Neural Network Works

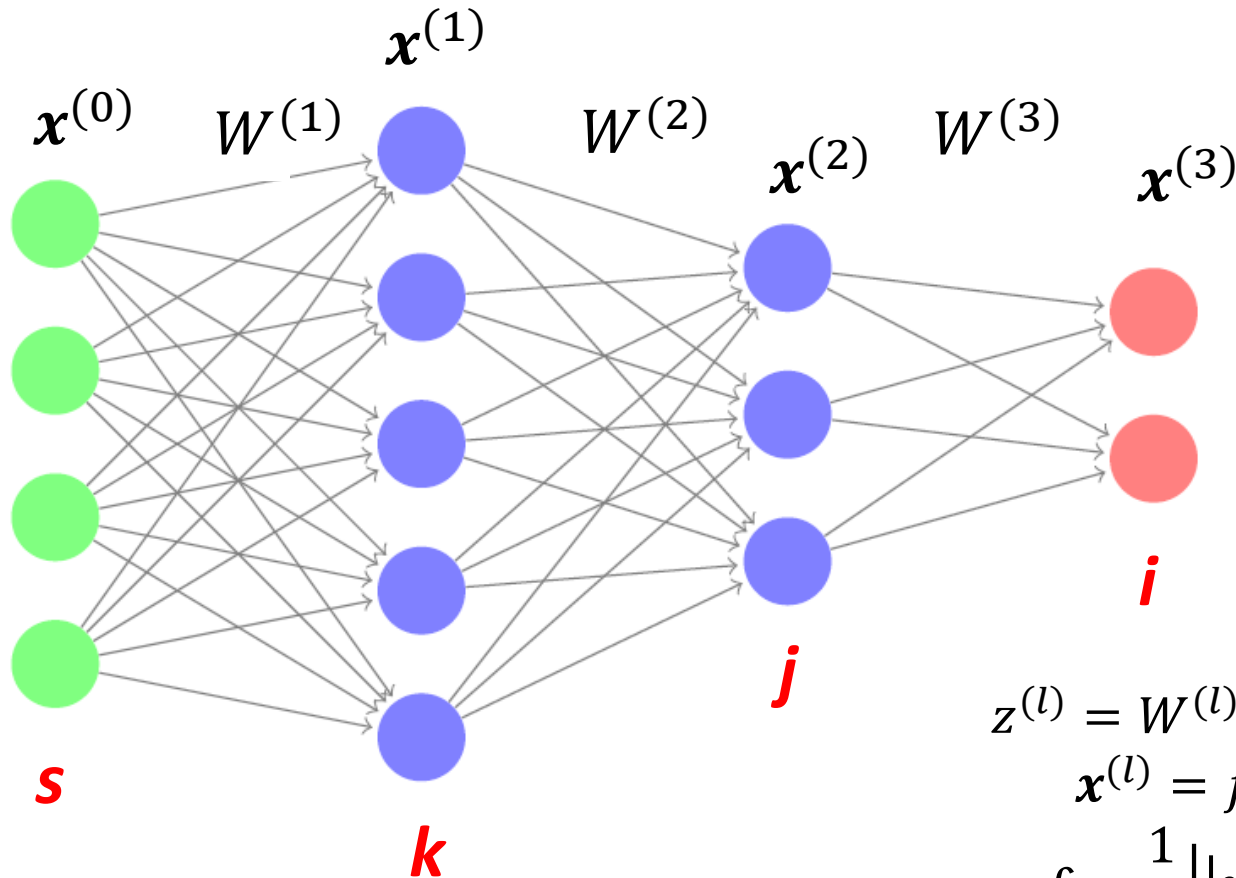
- The **inputs** to the network correspond to the attributes measured for each training tuple
- Inputs are fed simultaneously into the units making up the **input layer**
- They are then weighted and fed simultaneously to a **hidden layer**
 - The number of hidden layers is arbitrary
- The weighted outputs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



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

$$\mathbf{x}^{(l)} = f^{(l)}(z^{(l)})$$

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

Gradient for Layer 3 (Last Layer)

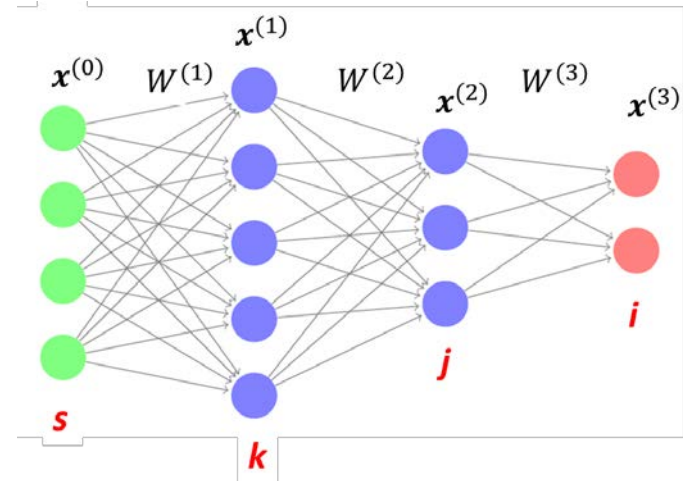
- Stochastic gradient for $W_{ij}^{(3)}$ and $b_i^{(3)}$

- Recall:*

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

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

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



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

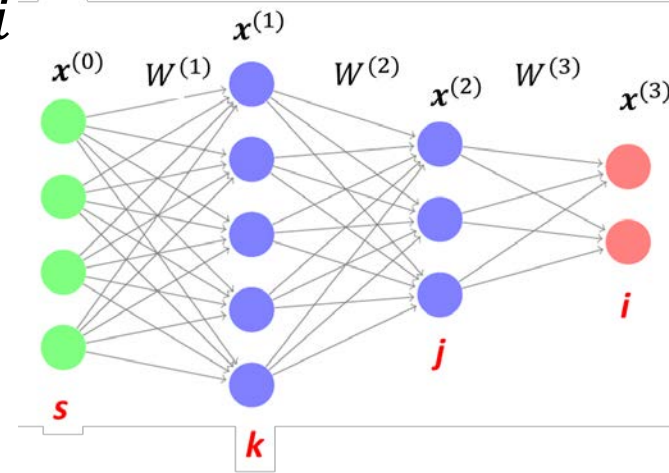
- $$\frac{\partial l}{\partial b_i^{(3)}} = \frac{\partial l}{\partial x_i^{(3)}} \frac{\partial x_i^{(3)}}{\partial z_i^{(3)}} \frac{\partial z_i^{(3)}}{\partial b_i^{(3)}} = -(y_i - x_i^{(3)}) \delta_i^{(3)} f'^{(3)}(z_i^{(3)})$$

Gradient for Layer 2

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

- Recall:*

- $\mathcal{L} = \frac{1}{2} \|\mathbf{y} - \mathbf{x}^{(3)}\|^2 = \frac{1}{2} \sum_i (y_i - x_i^{(3)})^2$
- $x_i^{(3)} = f^{(3)}(z_i^{(3)}); z_i^{(3)} = \sum_j W_{ij}^{(3)} x_j^{(2)} + b_i^{(3)}$
- $x_j^{(2)} = f^{(2)}(z_j^{(2)}); z_j^{(2)} = \sum_k W_{jk}^{(2)} x_k^{(1)} + b_j^{(2)}$

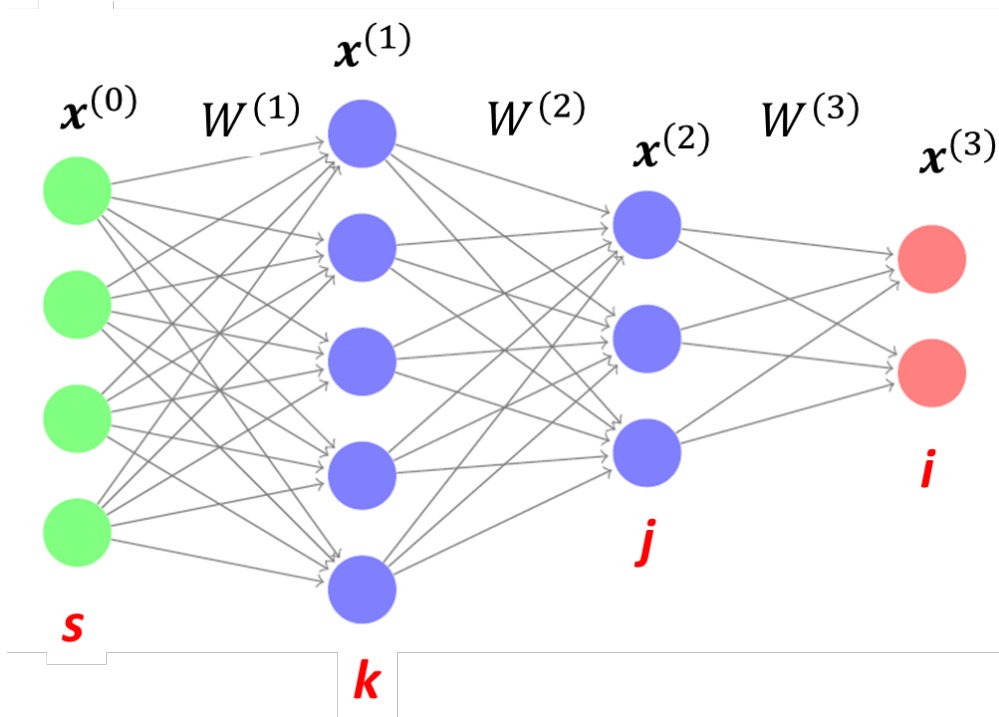


$$\begin{aligned}
 \bullet \frac{\partial \mathcal{L}}{\partial W_{jk}^{(2)}} &= \sum_i \frac{\partial \mathcal{L}}{\partial x_i^{(3)}} \frac{\partial x_i^{(3)}}{\partial z_i^{(3)}} \frac{\partial z_i^{(3)}}{\partial x_j^{(2)}} \frac{\partial x_j^{(2)}}{\partial z_j^{(2)}} \frac{\partial z_j^{(2)}}{\partial W_{jk}^{(2)}} \\
 &= \sum_i \underbrace{-(y_i - x_i^{(3)}) f'^{(3)}(z_i^{(3)})}_{\delta_i^{(3)}} \underbrace{W_{ij}^{(3)} f'^{(2)}(z_j^{(2)})}_{\delta_j^{(2)}} x_k^{(1)}
 \end{aligned}$$

Gradient for Layer 1

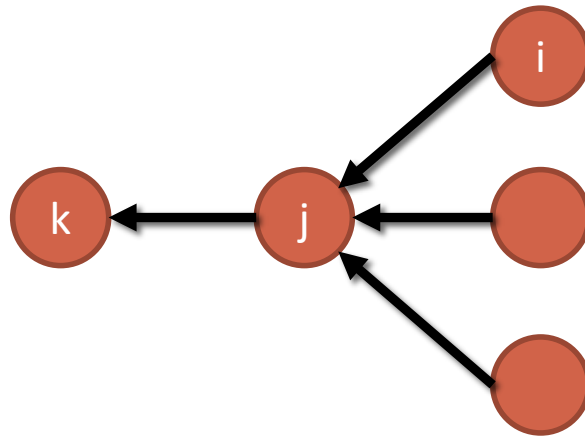
- Stochastic gradient for $W_{ks}^{(1)}$

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



Question: General form

- layer $l - 1 \rightarrow$ layer $l \rightarrow$ layer $l + 1$
 - Index: $k \rightarrow j \rightarrow i$



- $\delta_j^{(l)} = \sum_i \delta_i^{(l+1)} W_{ij}^{(l+1)} f'^{(l)}(z_j^{(l)})$
- $\frac{\partial \mathcal{L}}{\partial W_{jk}^{(l)}} = \delta_j^{(l)} x_k^{(l-1)}$

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 layer $l = 1: L$
 - Calculate $\mathbf{z}^{(l)} = W^{(l)}\mathbf{x}^{(l-1)} + \mathbf{b}^{(l)}$
 - Calculate $\mathbf{x}^{(l)} = f^{(l)}(\mathbf{z}^{(l)})$ (elementwise activation)
 - **Backpropagate the error** (by updating weights and biases)
 - Calculate $\boldsymbol{\delta}^{(L)} = \partial\mathcal{L}/\partial\mathbf{x}^{(L)} \circ f'^{(L)}(\mathbf{z}^{(L)})$
 - Update $W^{(L)}$ and $\mathbf{b}^{(L)}$ based on $\frac{\partial\mathcal{L}}{\partial W^{(L)}} = \boldsymbol{\delta}^{(L)}(\mathbf{x}^{(L-1)})^T$ and $\frac{\partial\mathcal{L}}{\partial \mathbf{b}^{(L)}} = \boldsymbol{\delta}^{(L)}$
 - For layer $l = L-1: 1$
 - Calculate $\boldsymbol{\delta}^{(l)} = (W^{(l+1)})^T \boldsymbol{\delta}^{(l+1)} \circ f'^{(l)}(\mathbf{z}^{(l)})$
 - Update $W^{(l)}$ and $\mathbf{b}^{(l)}$ based on $\frac{\partial\mathcal{L}}{\partial W^{(l)}} = \boldsymbol{\delta}^{(l)}(\mathbf{x}^{(l-1)})^T$ and $\frac{\partial\mathcal{L}}{\partial \mathbf{b}^{(l)}} = \boldsymbol{\delta}^{(l)}$
 - Terminating condition (convergence, max iteration, etc.)

slido



Audience Q&A Session

① Start presenting to display the audience questions on this slide.

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

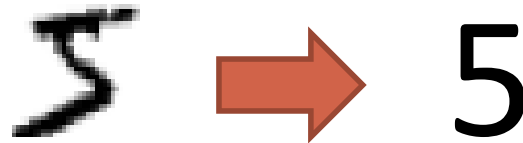
- High tolerance to noisy data
- Successful on an array of real-world data, e.g., hand-written letters
- Algorithms are inherently parallel
- Techniques have recently been developed for the extraction of rules from trained neural networks
- Deep neural network is powerful

Digits Recognition Example

- Obtain sequence of digits by segmentation

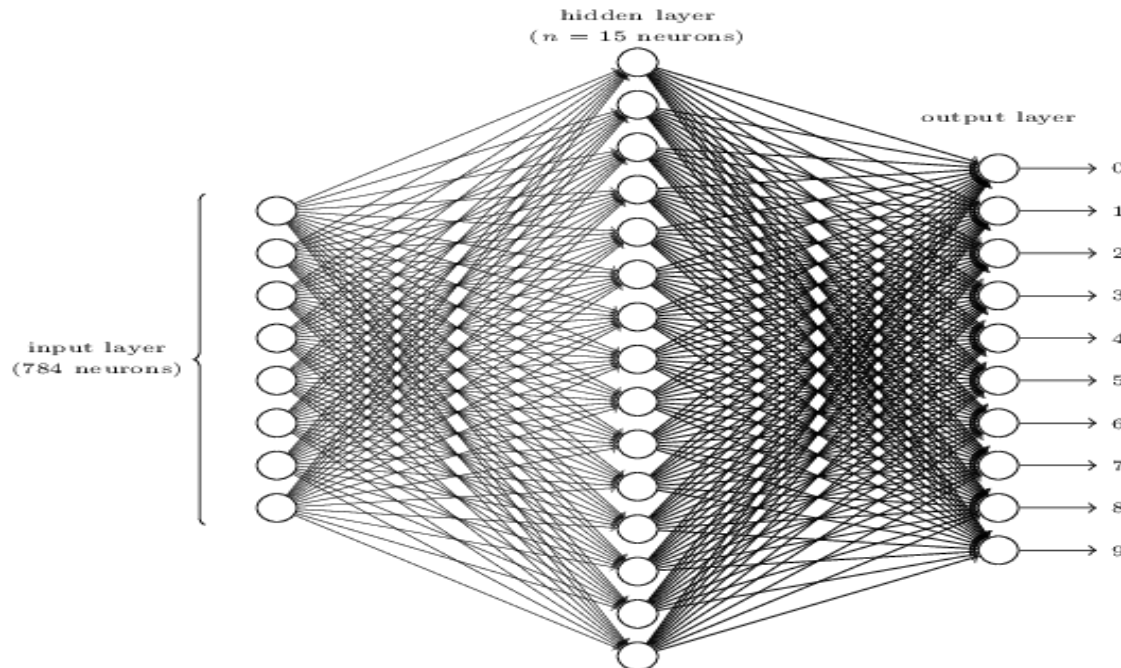


- Recognition (our focus)

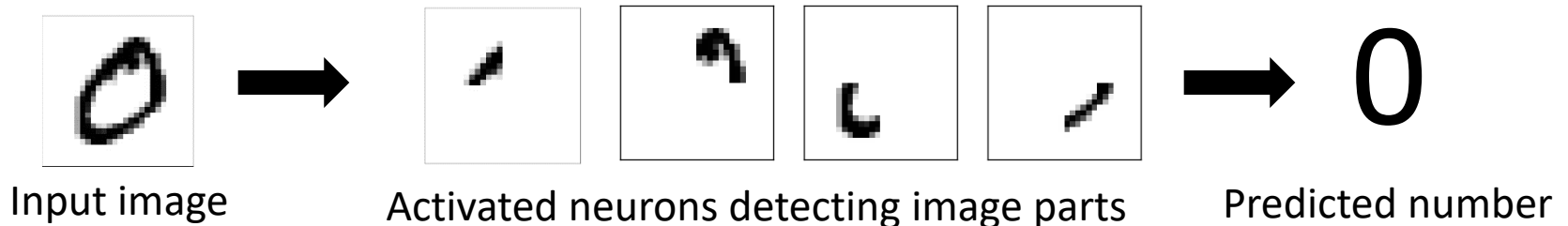


Digits Recognition Example


- The architecture of the used neural network



- What each neurons are doing?

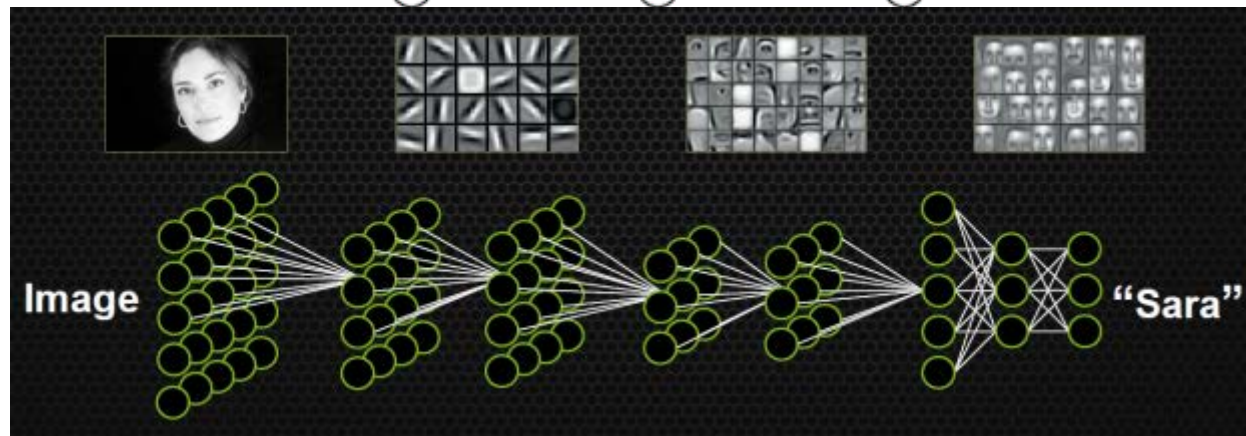
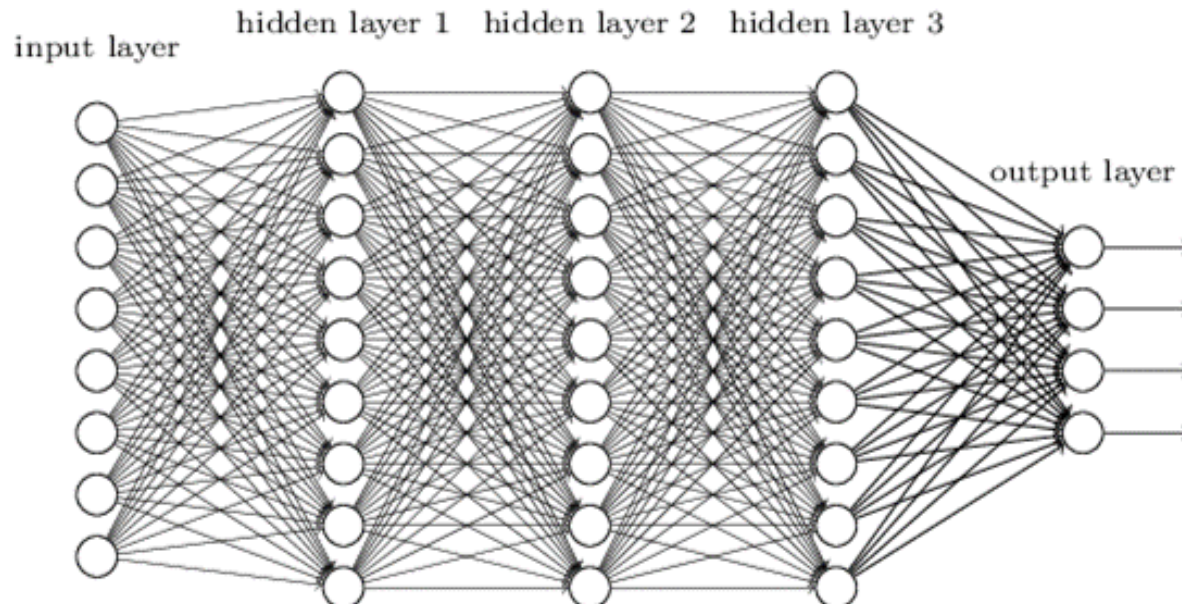


Neural Network

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

Towards Deep Learning

Deep neural network

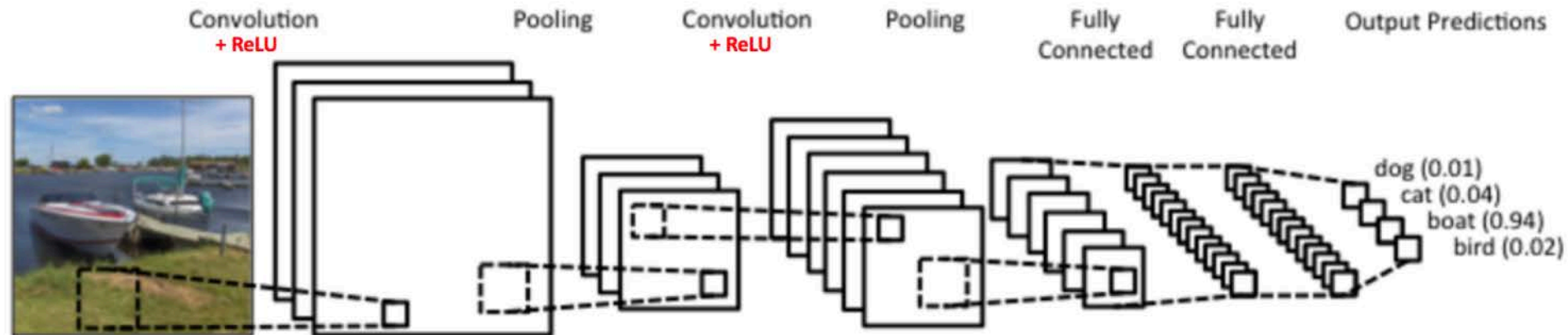


Popular Deep NNs

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

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

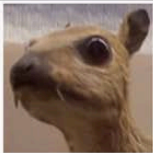



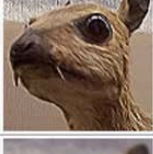
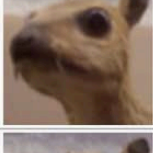
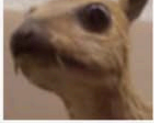
1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

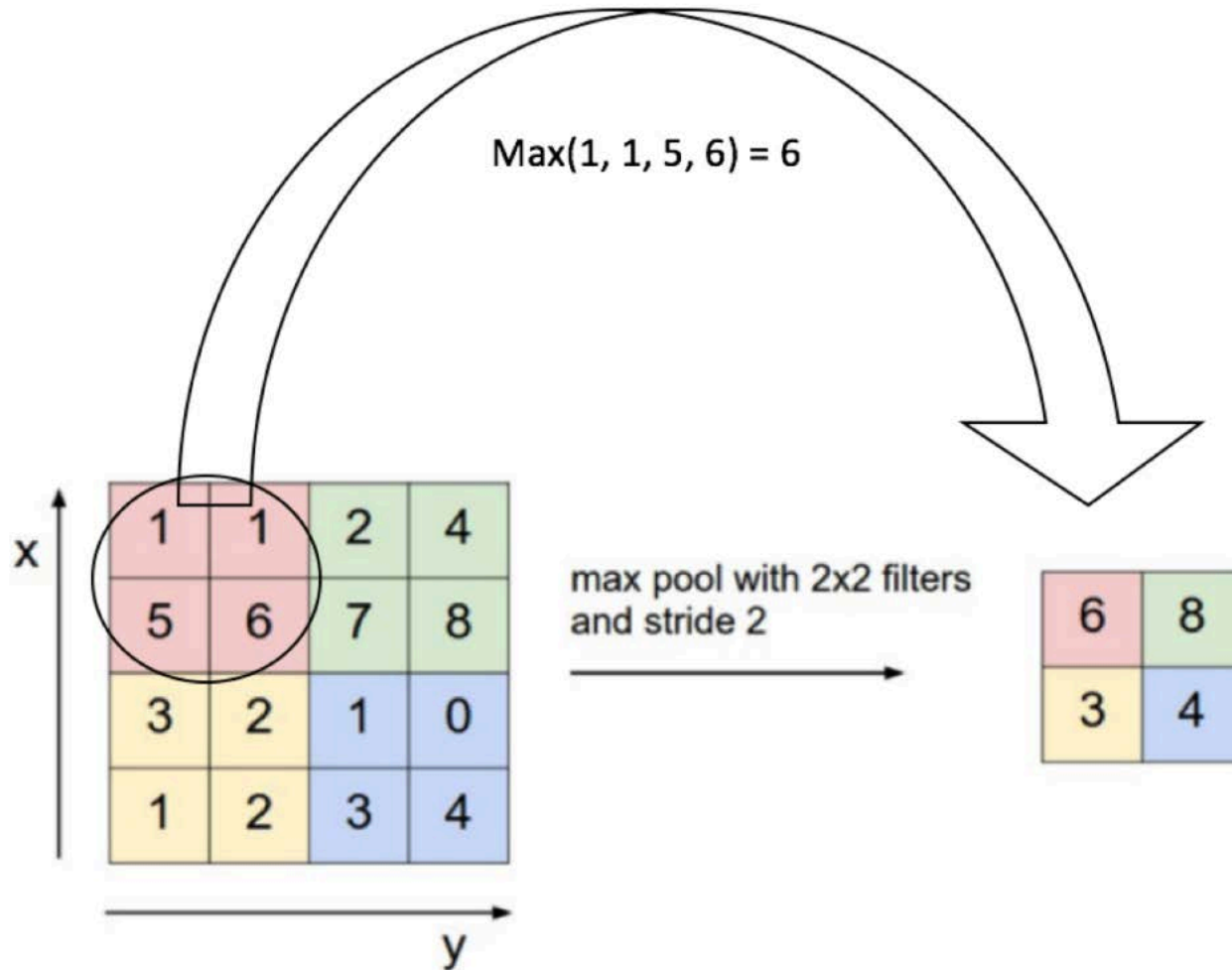
Convolved
Feature

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

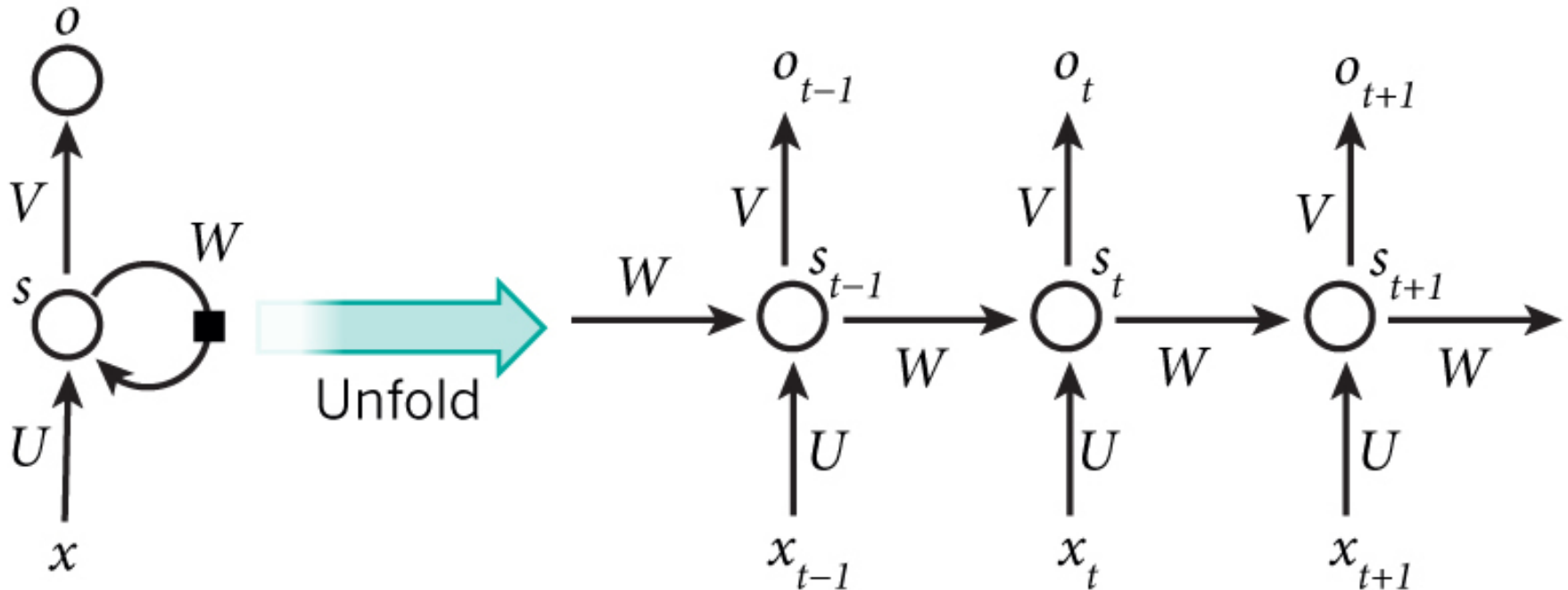


Rectified Feature Map

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

Recurrent Neural Network

- Model sequential information

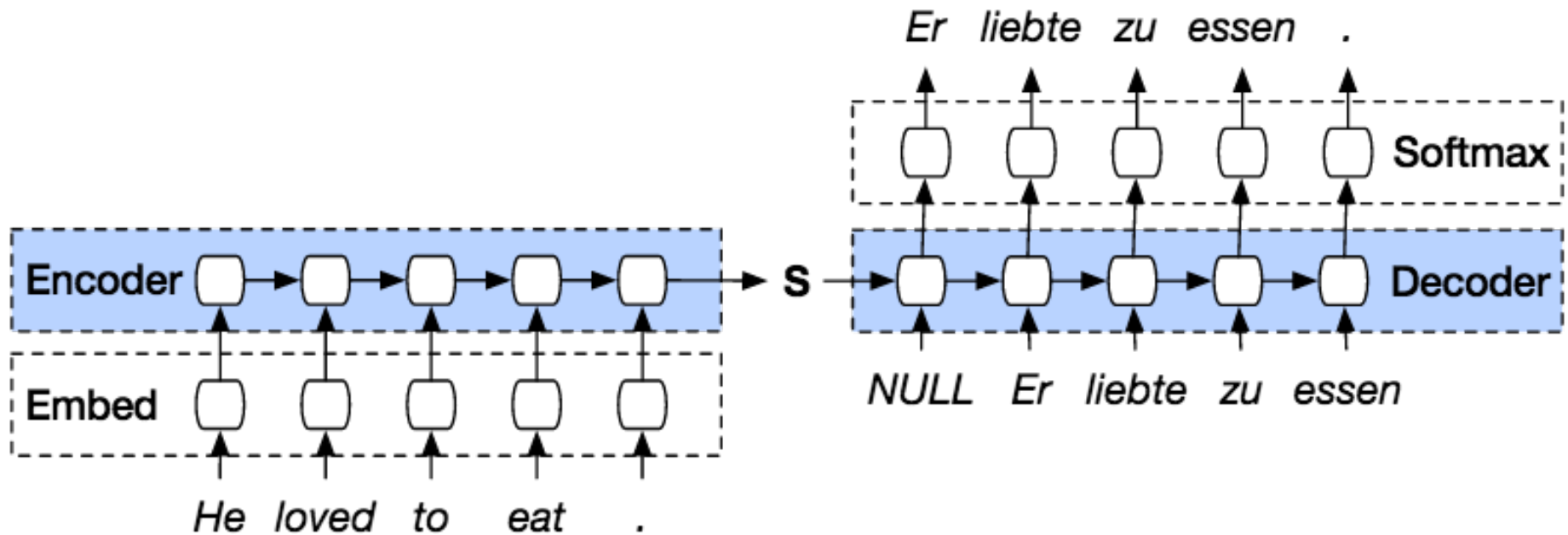


x_t : input vector at time t
 s_t : hidden state at time t
 o_t : output vector at time t

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

Application of Machine Translation

- Seq2Seq model



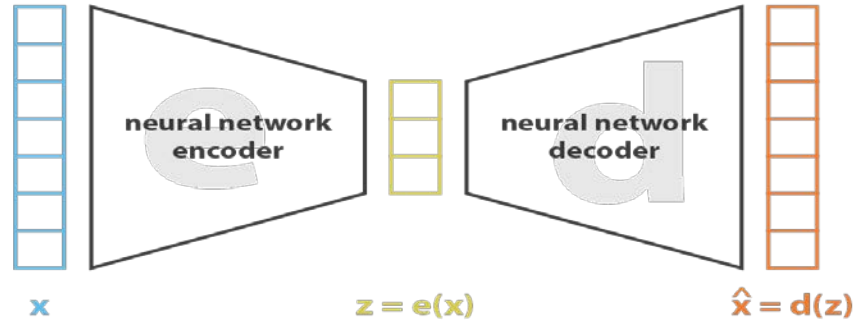
Variational Autoencoder

- Deep generative models
 - $\mathbf{z} \sim N(0, I)$
 - $\mathbf{x}|\mathbf{z} \sim N(G_{\theta}(\mathbf{z}), \eta I)$



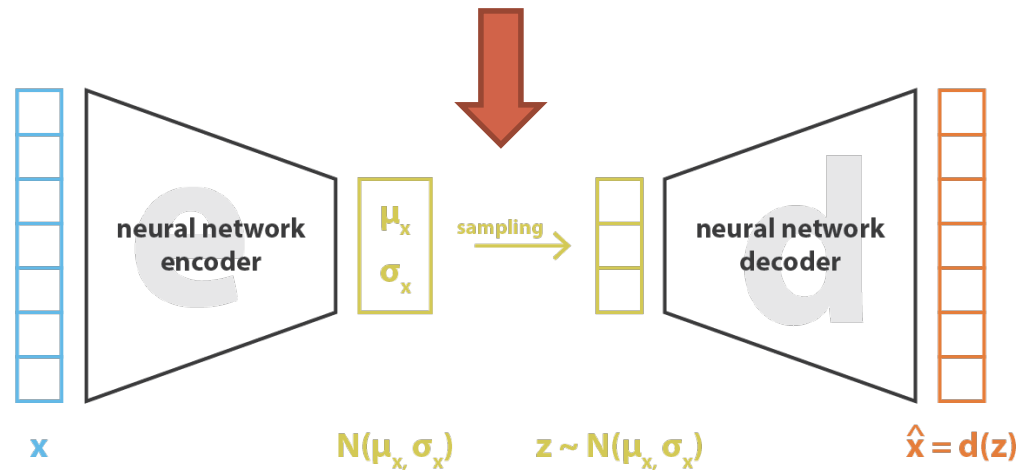
Variational Autoencoder

regular autoencoder



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

variational autoencoder



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

Other resources for variational autoencoder

- <https://www.youtube.com/watch?v=c27SHdQr4lw> by Paul Hand, 2020
- <https://arxiv.org/pdf/1906.02691.pdf>, by Kingma and Welling, 2019 (An introduction to variational autoencoders)

slido



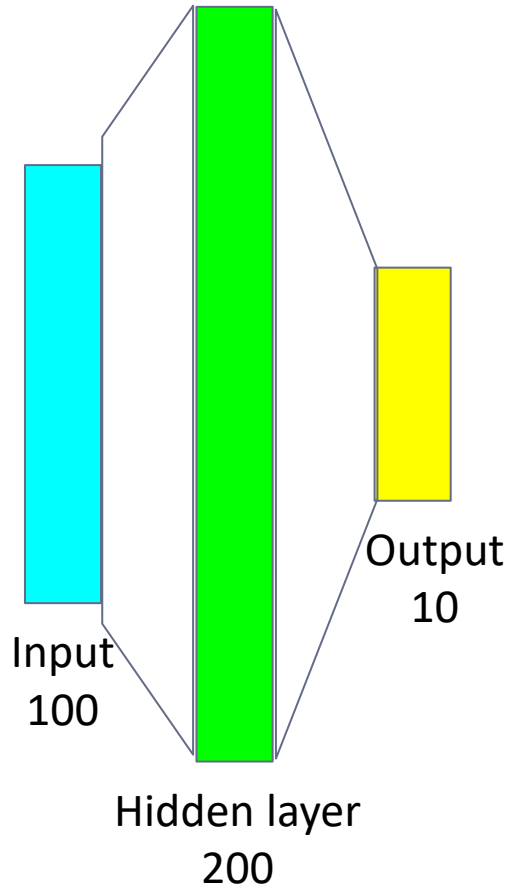
Audience Q&A Session

ⓘ Start presenting to display the audience questions on this slide.

Implementation

- Define architecture
- Define loss
- Training
 - Define optimizer
 - Define training schedule
 - Hyper parameter tuning
 - ...
- Test

Build a MLP with Pytorch



```
import torch
import torch.nn.functional as F
class TinyModel(torch.nn.Module):
    def __init__(self):
        # run the __init__ function of its super class
        super(TinyModel, self).__init__()
        # define each layer of the NN
        # torch.nn.Linear(input_dim, output_dim)
        self.linear1 = torch.nn.Linear(100, 200)
        self.activation = torch.nn.ReLU()
        self.linear2 = torch.nn.Linear(200, 10)
        self.softmax = torch.nn.Softmax()

    # define the forward pass.
    def forward(self, x):
        x = self.linear1(x)
        x = self.activation(x)
        x = self.linear2(x)
        x = self.softmax(x)
        return x
```

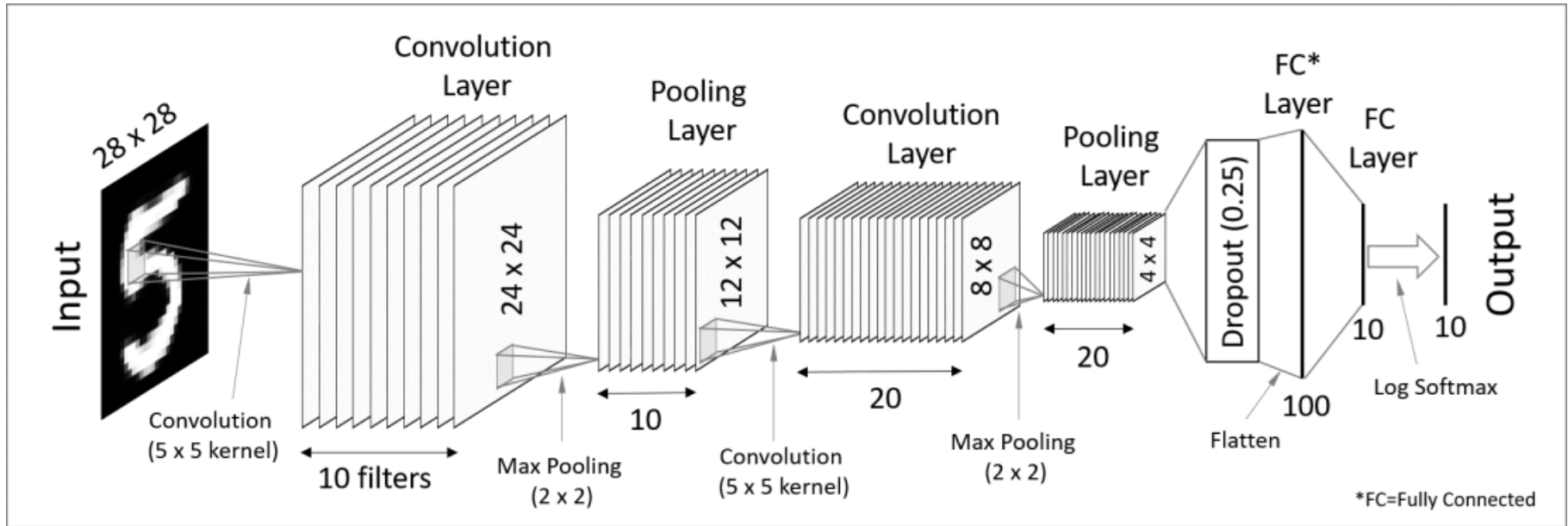
Build a CNN with Pytorch

Can you draw the architecture of this CNN?

```
class CNN(torch.nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        # activation function is omitted
        # assume the input image is 1*28*28
        # Conv2d(in_channels, out_channels, kernel_size)
        self.conv1 = torch.nn.Conv2d(1, 10, 5)
        # MaxPool2d(kernel_size, stride)
        self.pool1 = torch.nn.MaxPool2d(2, 2)
        self.conv2 = torch.nn.Conv2d(10, 20, 5)
        self.pool2 = torch.nn.MaxPool2d(2, 2)
        self.dropout = torch.nn.Dropout(p=0.25)
        # now the feature shape is 20*4*4
        self.fc1 = torch.nn.Linear(20*4*4, 100)
        self.fc2 = torch.nn.Linear(100, 10)

    def forward(self, x):
        # Omitted
```

Build a CNN with Pytorch



Train the Neural Network with Pytorch

Do the forward pass and compute the loss function

Zero out the previous gradients

Do the backward propagation

Do the optimizer step

```
# Create a TinyModel instance
model = TinyModel()
# Initialize the loss function
loss_fn = nn.CrossEntropyLoss()
# Initialize the optimizer
learning_rate = 1e-3
batch_size = 64
train_dataloader = DataLoader(training_data, batch_size=batch_size)
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
# Define the train loop to optimize our model
def train_loop(dataloader, model, loss_fn, optimizer):
    for (X, y) in dataloader:
        # Compute prediction and loss
        pred = model(X)
        loss = loss_fn(pred, y)
        # Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```


Test the Neural Network with Pytorch

```
def test_loop(dataloader, model, loss_fn):
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    test_loss, correct = 0, 0

    with torch.no_grad(): # no need to compute gradients
        for X, y in dataloader:
            pred = model(X)
            # item function return a float number
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()

    test_loss /= num_batches
    correct /= size
    print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, \
Avg loss: {test_loss:>8f} \n")
```

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

slido



Audience Q&A Session

① Start presenting to display the audience questions on this slide.

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

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