CS247: ADVANCED DATA MINING

04: Basics: Neural Network and Deep Learning

Instructor: Yizhou Sun

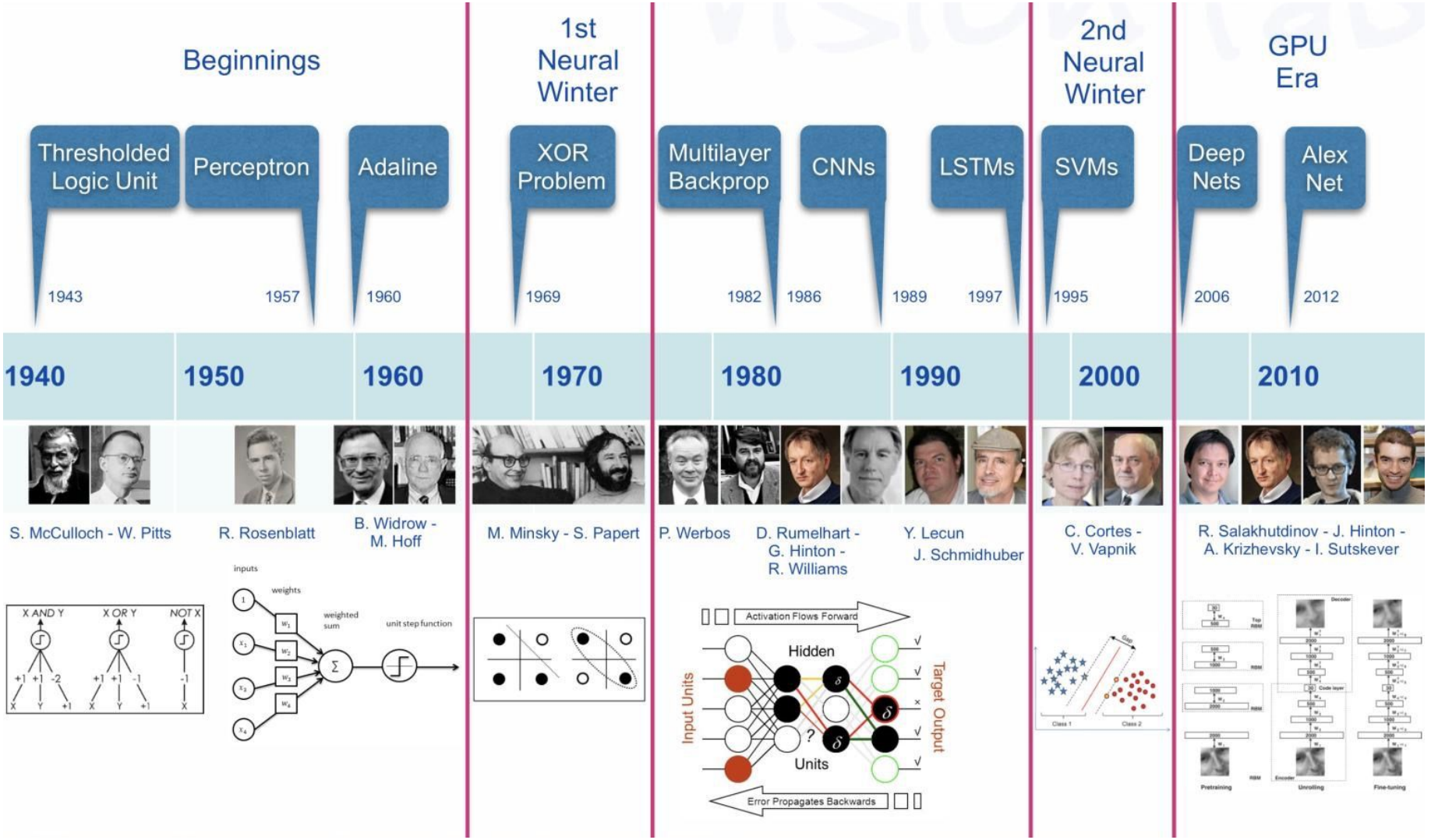
yzsun@cs.ucla.edu

January 29, 2024

Neural Network

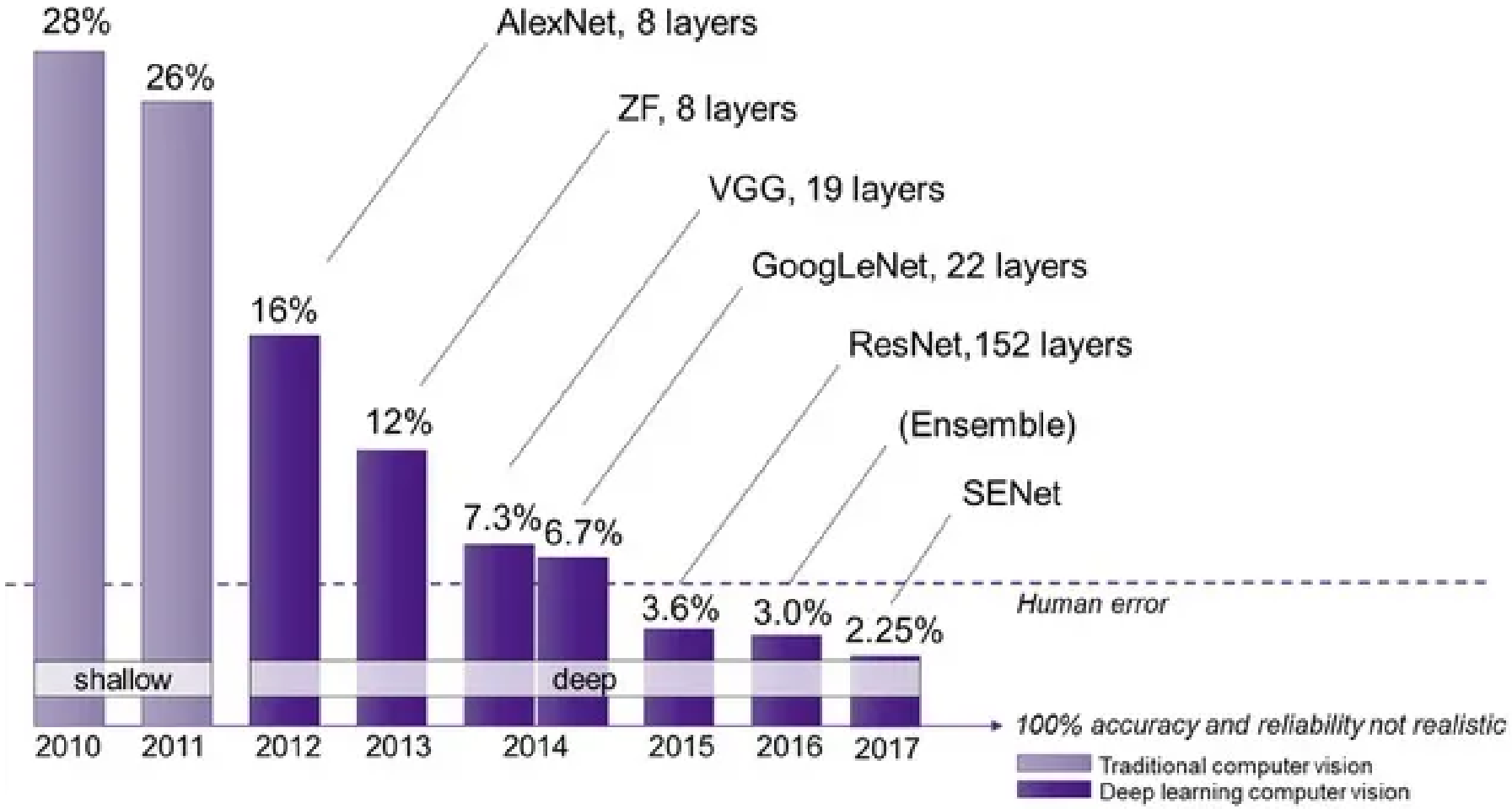
- Introduction 
- Connection to Shallow Machine Learning Algorithms
- Multi-Layer Feed-Forward Neural Network
- Deep Learning
- Summary

A little bit of History



https://i.pinimg.com/originals/6a/f0/3c/6af03c5026bb680ebe6d8db4bdbb8428.jpg

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

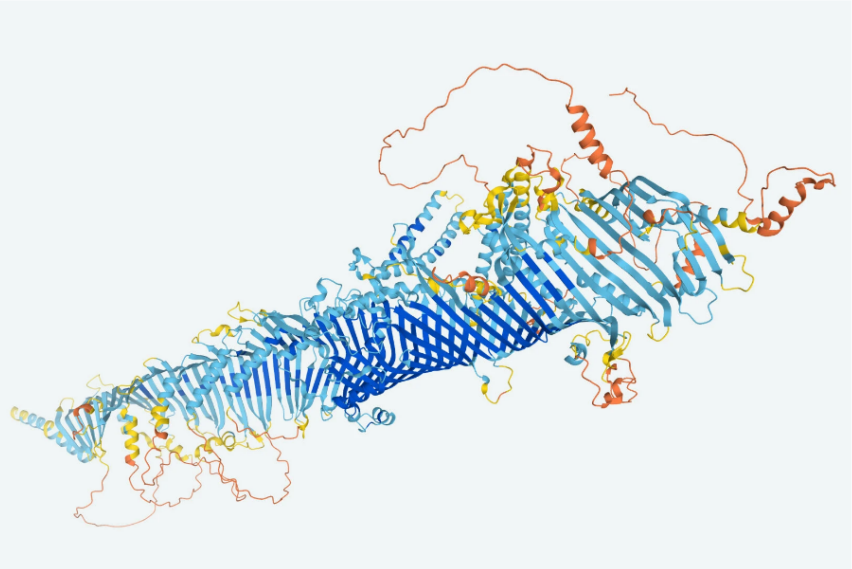
SEOUL, South Korea — Computer, one. Human, zero.

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

OpenAI: Codex

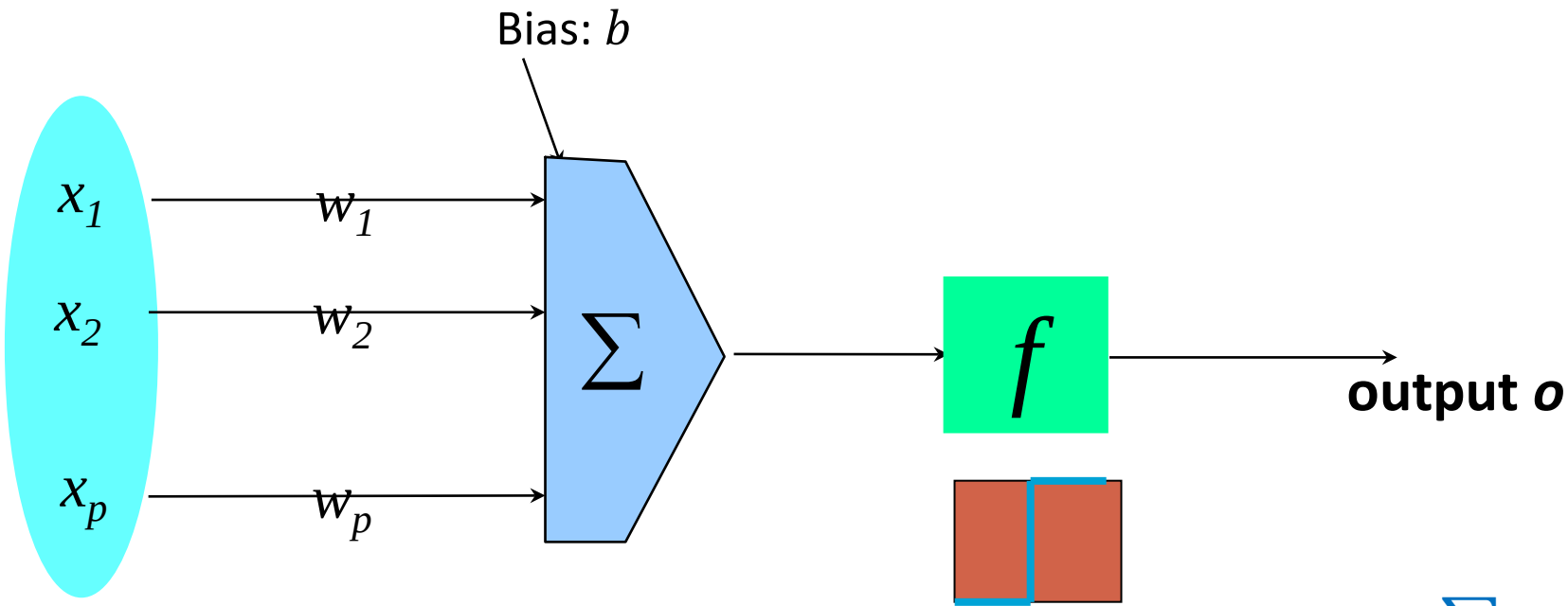
Make it be the size of the rocketship times 0.75

```
  text.style.left =
rocketship.offsetLeft + 'px';
  text.style.top =
rocketship.offsetTop + 'px';

document.body.appendChild(text
);
  xSpeed = 20;
  setTimeout(function() {
    xSpeed = 5;

document.body.removeChild(text
);
  }, 250);
};



/* Now add an image of an
asteroid:
https://d.newsweek.com/en/full
/1721338/asteroid.jpg?
w=1600&h=1600&q=88&f=9d82d35c9
de96a82b3fcaf7705eb325b */
var asteroid =
document.createElement('img');
asteroid.src =
'https://d.newsweek.com/en/ful
l/1721338/asteroid.jpg?
w=1600&h=1600&q=88&f=9d82d35c9
de96a82b3fcaf7705eb325b';
document.body.appendChild(aste
roid);
```

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



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

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

Important Concepts

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

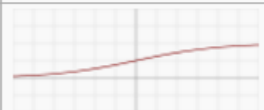
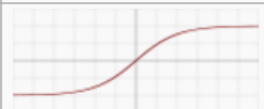
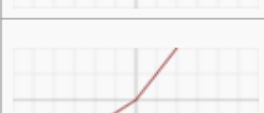
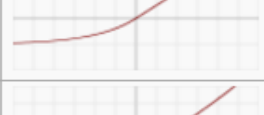
Architecture

- 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		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x \neq 0 \\ ? & \text{for} \quad x = 0 \end{cases}$
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)		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Q&A

Loss Functions

- How good are the outputs compared with the labels (target)?
  - Empirical risk
    - $\mathcal{L}(\boldsymbol{w}) = \frac{1}{n}\sum_i l\left(y^{(i)}, \hat{y}^{(i)}\right)$, 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

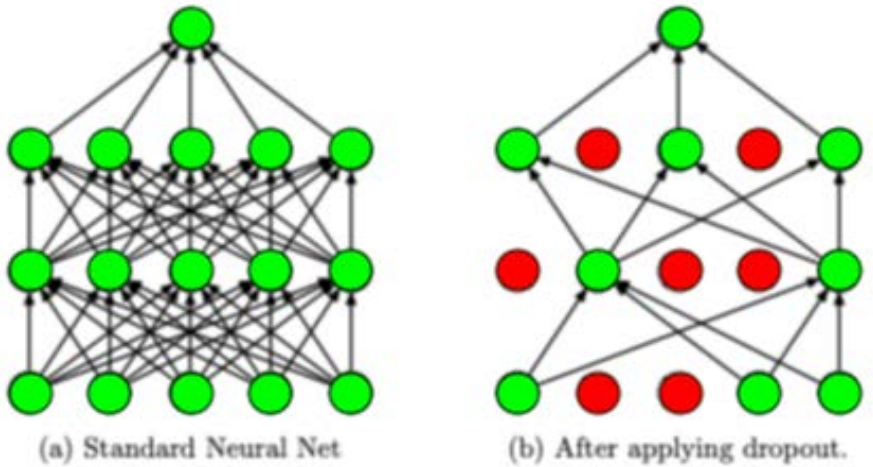
- Squared error
  - $l(y, \hat{y}) = (y - \hat{y})^2$
- (Binary) cross entropy loss
  - $l(y, \hat{y}) = -ylog\hat{y} - (1 - y)log(1 - \hat{y})$
  - $y \in \{0,1\}, \hat{y} \in [0,1]$
- Hinge loss
  - $l(y, \hat{y}) = \max(0,1 - y\hat{y})$
  - $y \in \{-1,1\}, \hat{y} \in (-\infty, +\infty)$

Optimization

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

Regularization

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



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

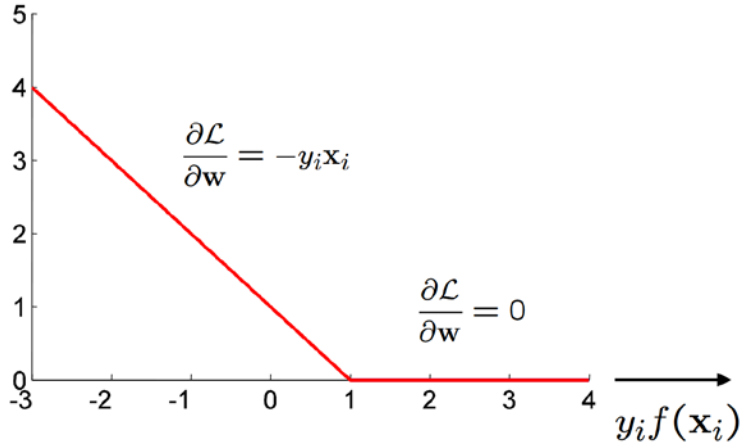
- Architecture:
  - A single neuron
- Activation function
  - Training: identity function
  - Inference: sign function/step function
- Loss function
  - $l(y, \hat{y}) = \max(0, -y\hat{y})$
- Optimization
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta y^{(i)} \boldsymbol{x}^{(i)}$, for a misclassified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

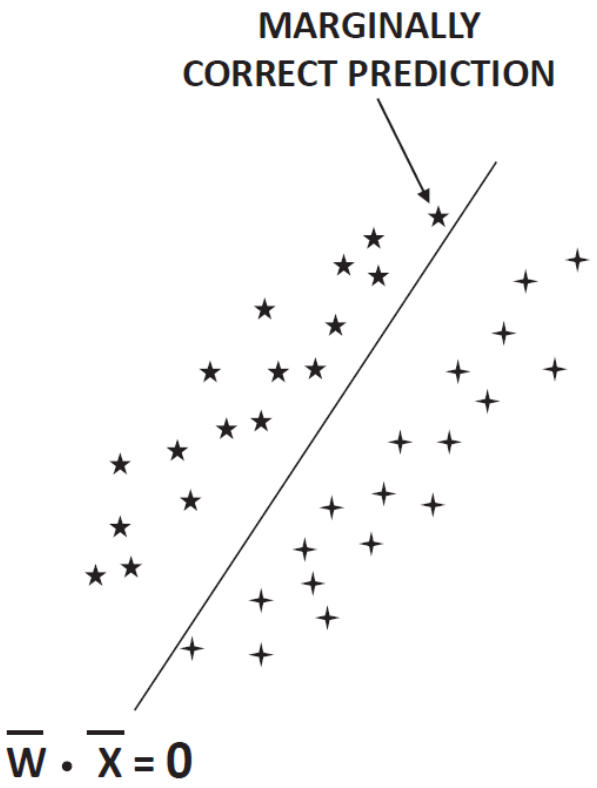
Linear SVM

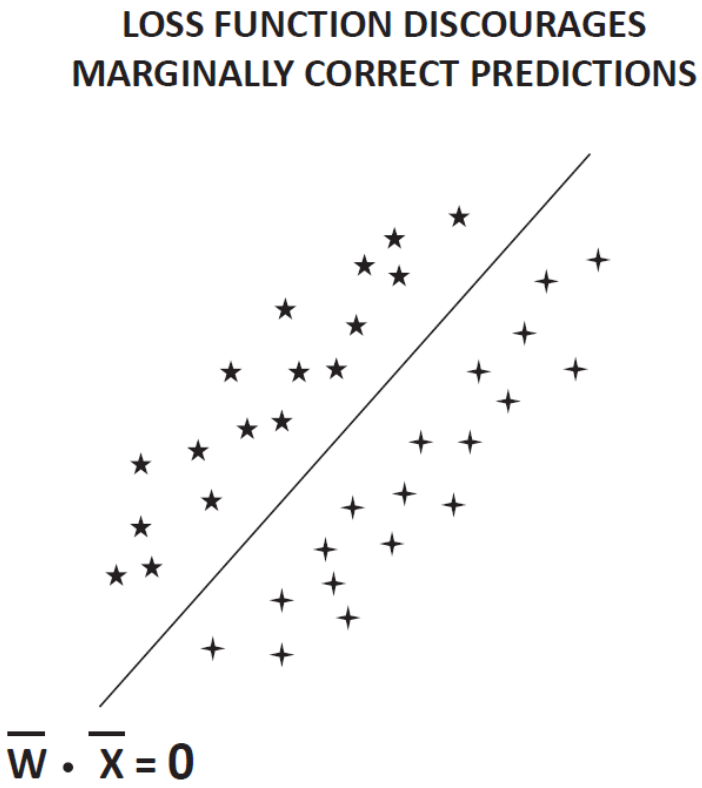
- Architecture:
  - A single neuron
- Activation function
  - Training: identity function
  - Inference: sign function/step function
- Loss function
  - $l(y, \hat{y}) = \max(0, 1 - y\hat{y})$
- Regularization
  - L2, i.e., $\frac{1}{2}\lambda||\boldsymbol{w}||^2$
- Optimization
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta(\lambda w - y_i \boldsymbol{x}_i)$, 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
  - 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
  - $l(y, \hat{y}) = -ylog\hat{y} - (1-y)log(1-\hat{y})$
  - Note $\hat{y}$ is the predicted probability of taking class 1
- Optimization
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta\left(y_i - \sigma(w^T \boldsymbol{x}_i)\right)\boldsymbol{x}_i$, for a training data point $(\boldsymbol{x}_i, y_i)$
  - η: learning rate

Question

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

Question

- How to view multivariate logistic regression in the neural network setting?

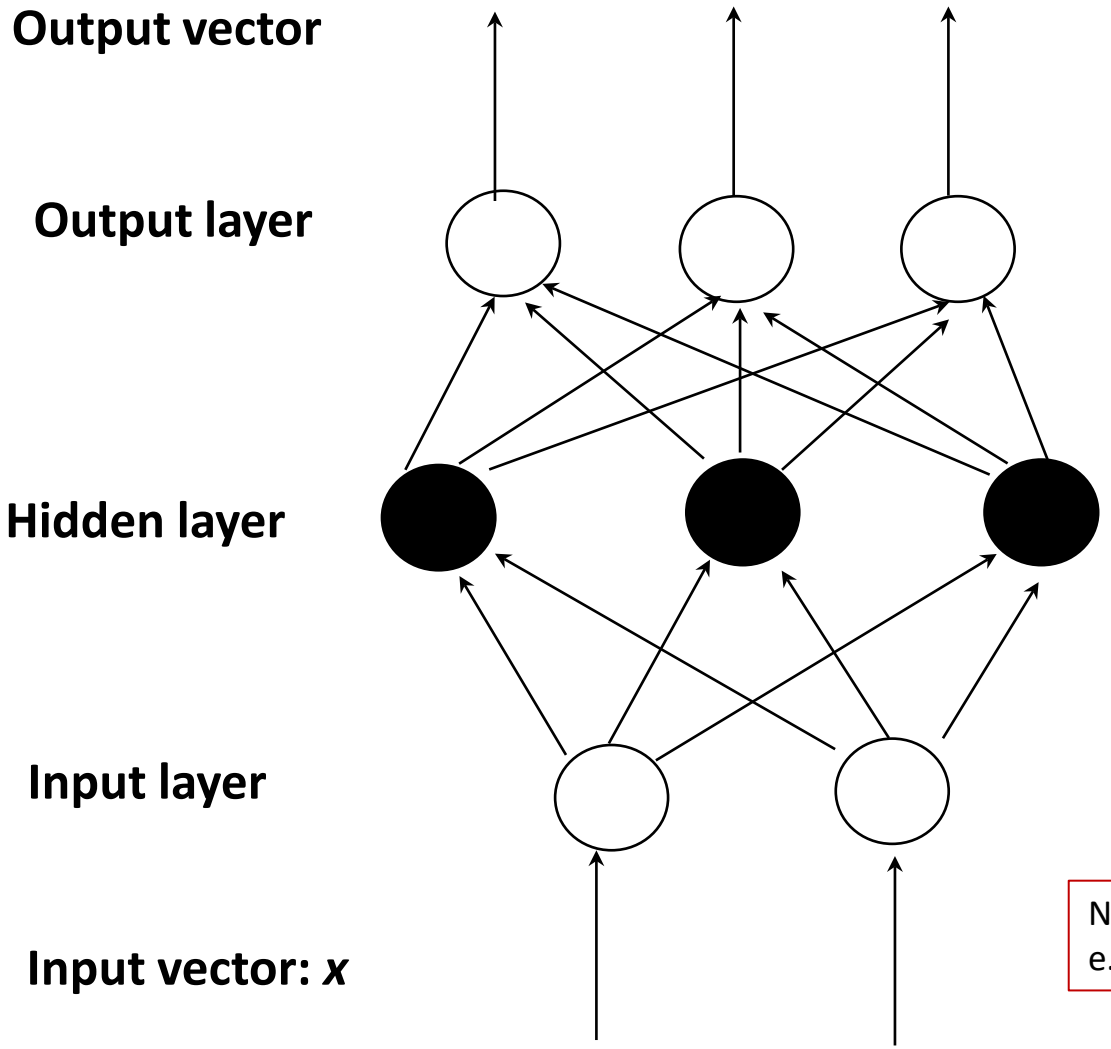
Neural Network

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

Announcement

- Midterm:
  - 2/21 in class for in person students
  - 2/10 for online students (unchanged)

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**"

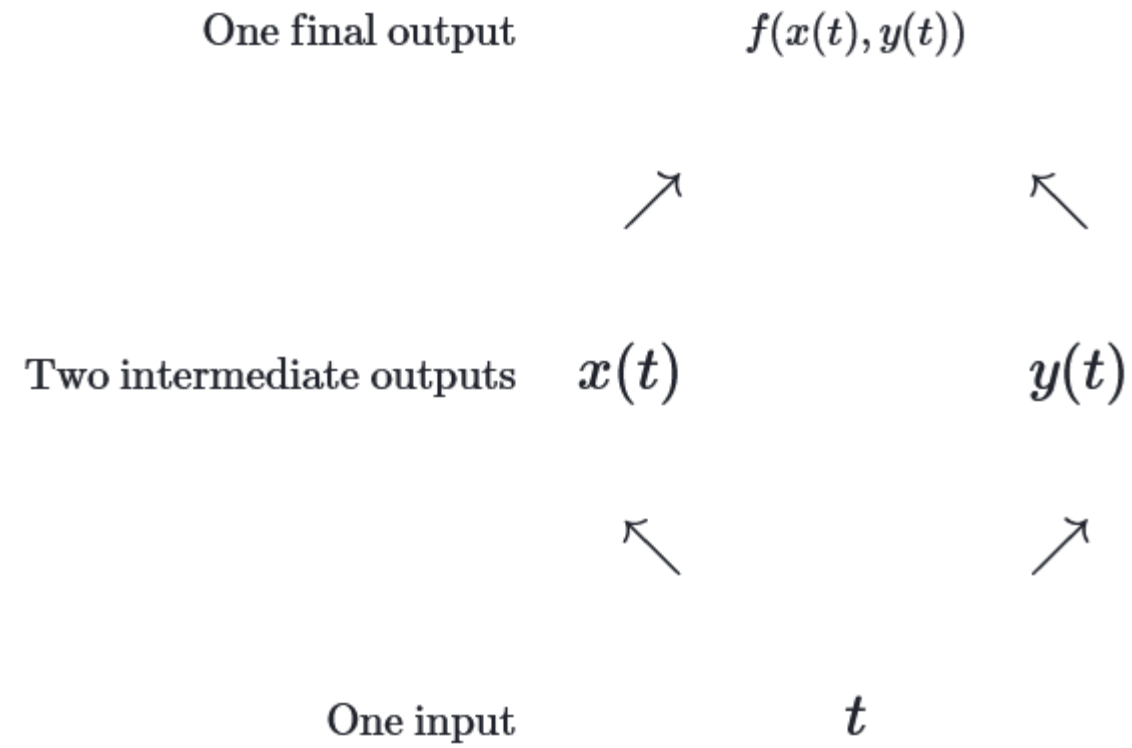
Recap: Chain Rule

- Single variable chain rule

$$\frac{d}{dt} f(g(t)) = \frac{df}{dg}\frac{dg}{dt} = f'(g(t))g'(t)$$

- Multiple variable chain rule

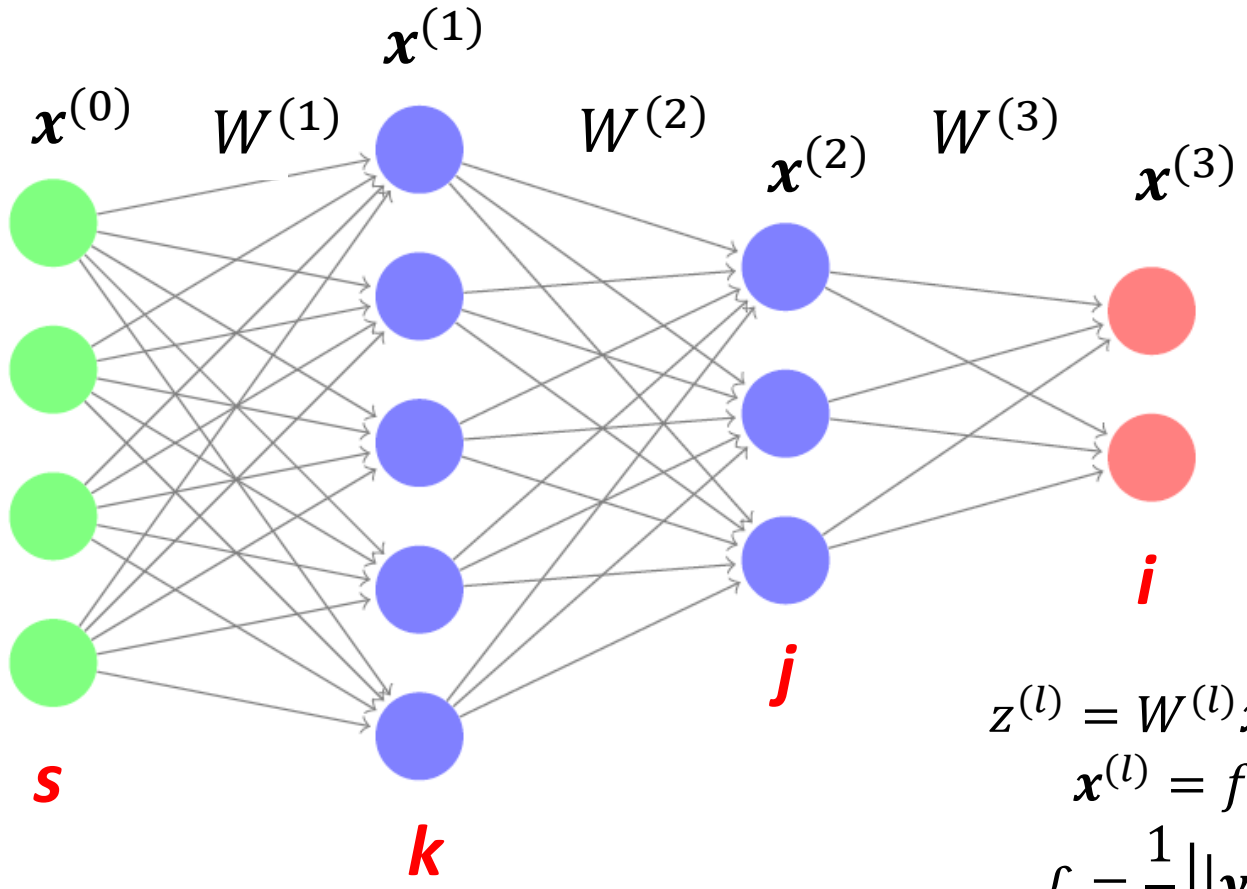
$$\underbrace{\frac{d}{dt} f(x(t), y(t))}_{\text{Derivative of composition function}} = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt}$$



https://www.khanacademy.org/math/multivariable-calculus/multivariable-derivatives/differentiating-vector-valued-functions/a/multivariable-chain-rule-simple-version

Example

- Loss function: $\mathcal{L} = \frac{1}{2}||\boldsymbol{y} - \hat{\boldsymbol{y}}||^2$



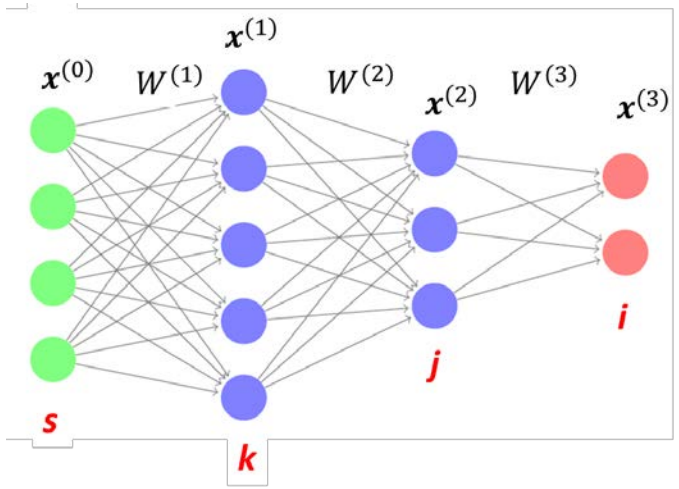
$$z^{(l)} = W^{(l)}\boldsymbol{x}^{(l-1)} + b^{(l)}$$

$$\boldsymbol{x}^{(l)} = f^{(l)}\left(z^{(l)}\right)$$

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

Gradient for Layer 3 (Last Layer)

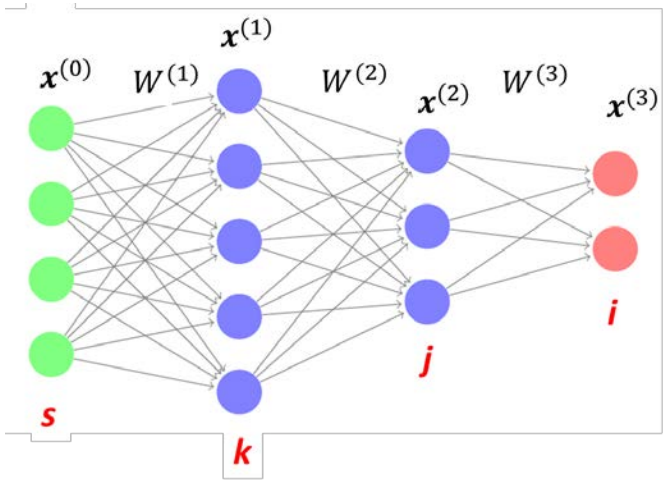
- Stochastic gradient for $W_{ij}^{(3)}$ and $b_i^{(3)}$
  - *Recall:*
    - $\mathcal{L} = \frac{1}{2}\left|\left|\boldsymbol{y} - \boldsymbol{x}^{(3)}\right|\right|^2 = \frac{1}{2}\sum_i \left(y_i - x_i^{(3)}\right)^2$
    - $x_i^{(3)} = f^{(3)}(z_i^{(3)})$
    - $z_i^{(3)} = \sum_j W_{ij}^{(3)} x_j^{(2)}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



- $\frac{\partial \mathcal{L}}{\partial W_{ij}^{(3)}} = \frac{\partial \mathcal{L}}{\partial x_i^{(3)}} \frac{\partial x_i^{(3)}}{\partial z_i^{(3)}} \frac{\partial z_i^{(3)}}{\partial W_{ij}^{(3)}} = \underbrace{-(y_i - x_i^{(3)})\mathrm{f}'^{(3)}\left(\mathrm{z}_\mathrm{i}^{(3)}\right)}_{\boldsymbol{\delta_i^{(3)}}} \mathrm{x}_j^{(2)}$
- $\frac{\partial \mathcal{L}}{\partial b_i^{(3)}} = \frac{\partial \mathcal{L}}{\partial x_i^{(3)}} \frac{\partial x_i^{(3)}}{\partial z_i^{(3)}} \frac{\partial z_i^{(3)}}{\partial b_i^{(3)}} = -(y_i - x_i^{(3)})\mathrm{f}'^{(3)}\left(\mathrm{z}_\mathrm{i}^{(3)}\right)$

Gradient for Layer 2

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

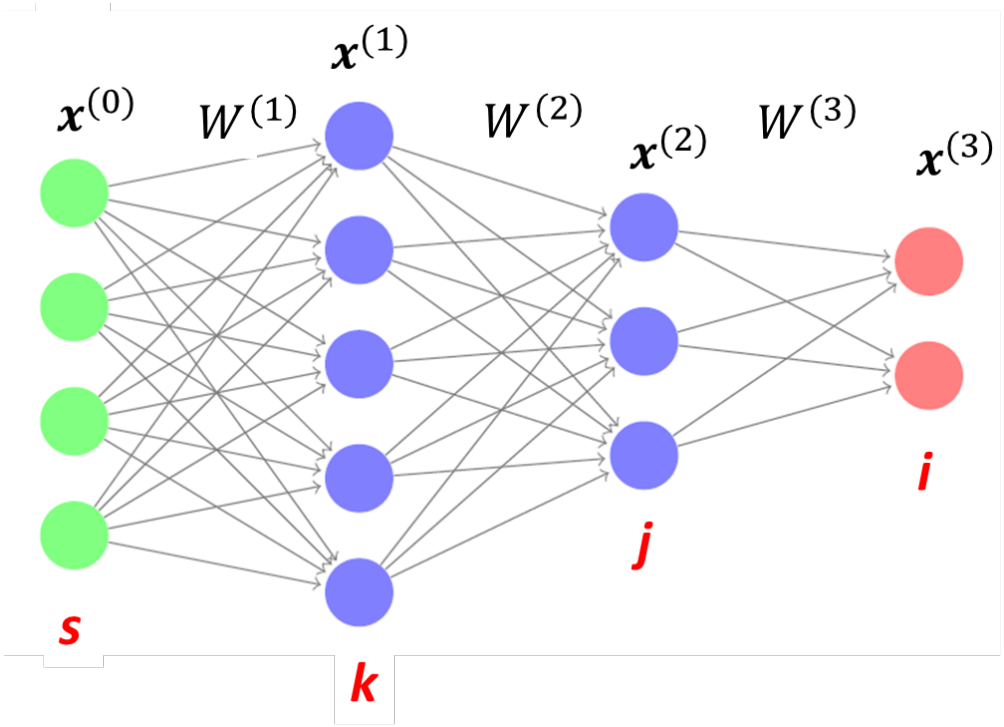


- $$\frac{\partial \mathcal{L}}{\partial W_{jk}^{(2)}} = \sum_i \frac{\partial \mathcal{L}}{\partial x_i^{(3)}} \frac{\partial x_i^{(3)}}{\partial z_i^{(3)}} \frac{\partial z_i^{(3)}}{\partial x_j^{(2)}} \frac{\partial x_j^{(2)}}{\partial z_j^{(2)}} \frac{\partial z_j^{(2)}}{\partial W_{jk}^{(2)}}$$

$$= \sum_i \underbrace{\underbrace{-(y_i - x_i^{(3)})\mathrm{f}'^{(3)}\left(\mathrm{z}_\mathrm{i}^{(3)}\right)}_{\delta_i^{(3)}} \mathrm{w}_{ij}^{(3)}\ \mathrm{f}'^{(2)}\left(\mathrm{z}_\mathrm{j}^{(2)}\right) x_k^{(1)}}_{\delta_j^{(2)}}$$

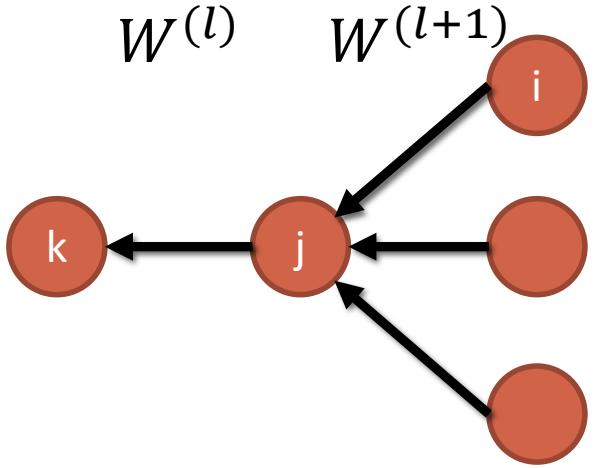
Gradient for Layer 1

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



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

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

$\circ$: Hadamard product (element wise product)

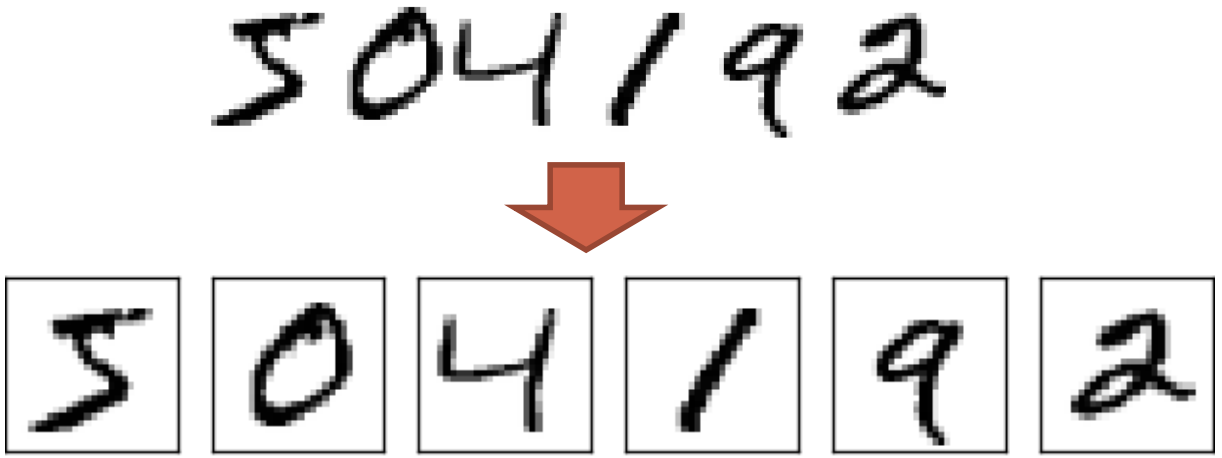
Q&A

Neural Network as a Classifier

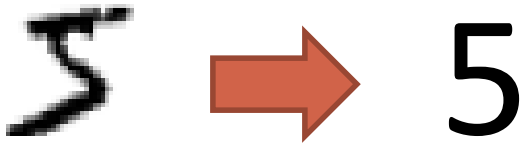
- Weakness
  - Long training time
  - Require a number of hyper-parameters typically best determined empirically, e.g., the network topology or "structure."
  - Poor interpretability: Difficult to interpret the symbolic meaning behind the learned weights and of "hidden units" in the network
- Strength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

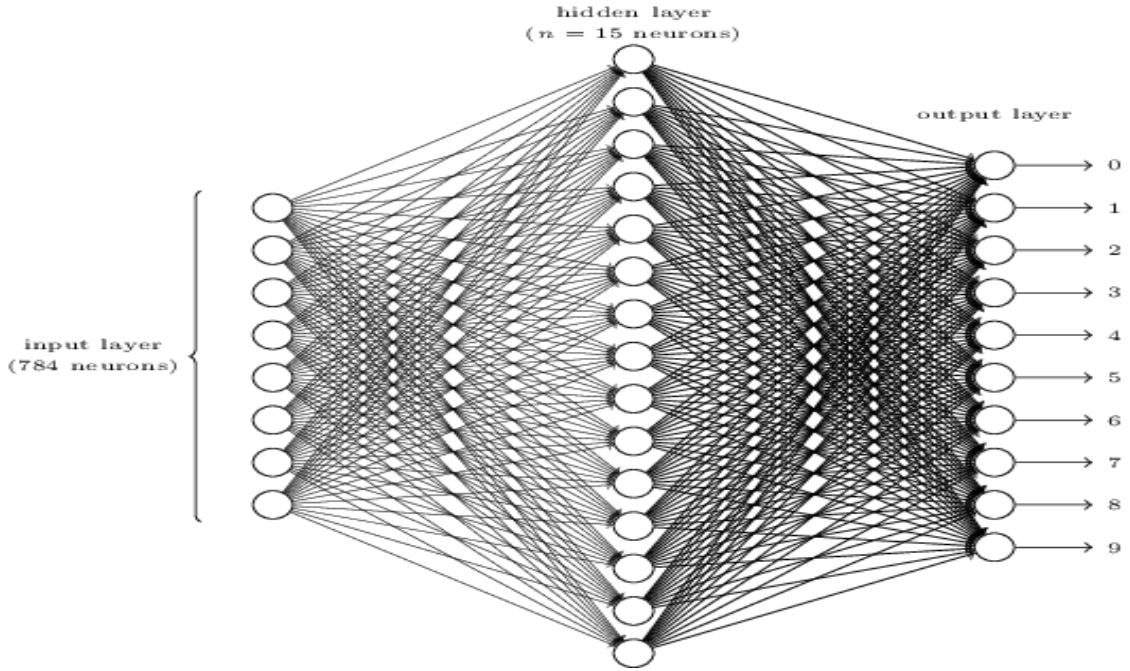


- Recognition (our focus)

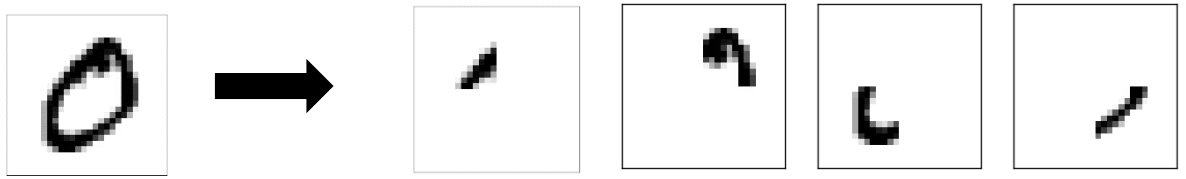


Digits Recognition Example

- The architecture of the used neural network



- What each neurons are doing?



Input image

Activated neurons detecting image parts

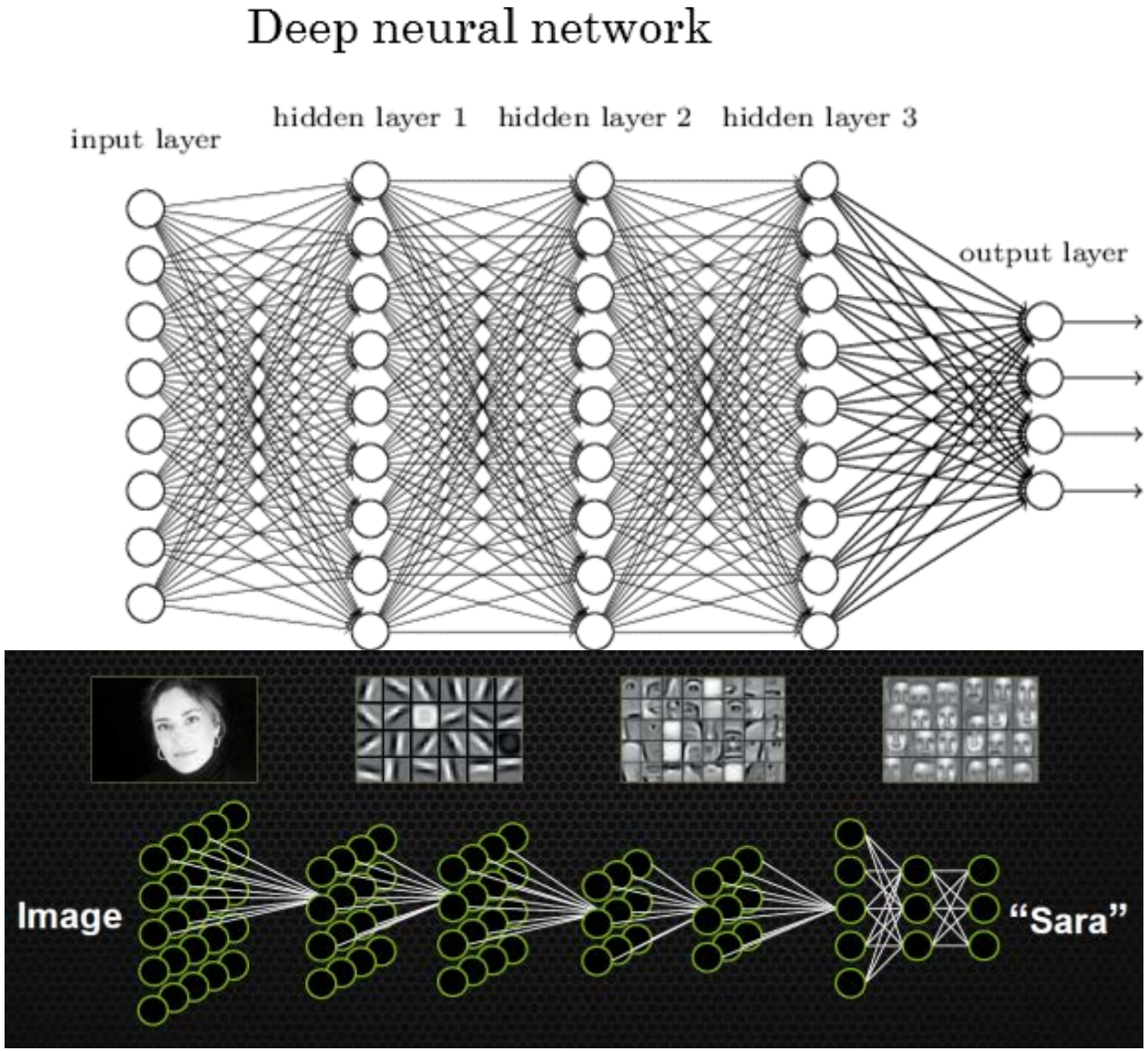
Predicted number

0

Neural Network

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

Towards Deep Learning

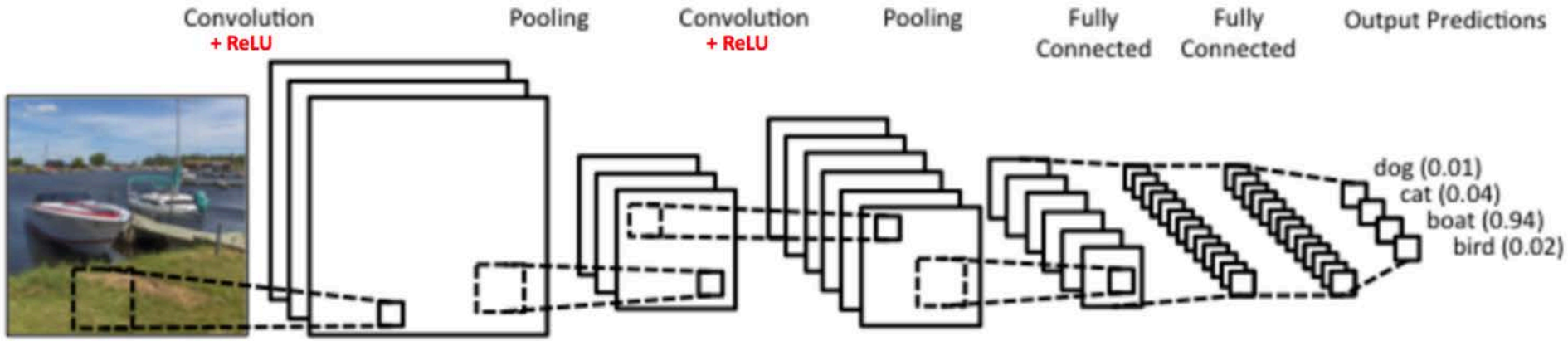


Popular Deep NNs

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

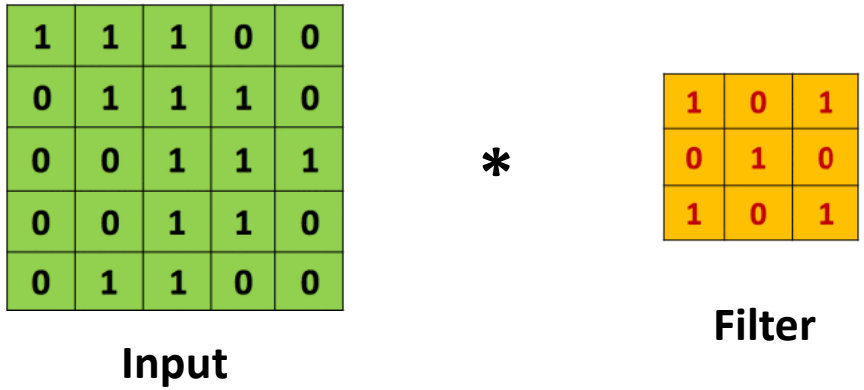
Convolutional Neural Network

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

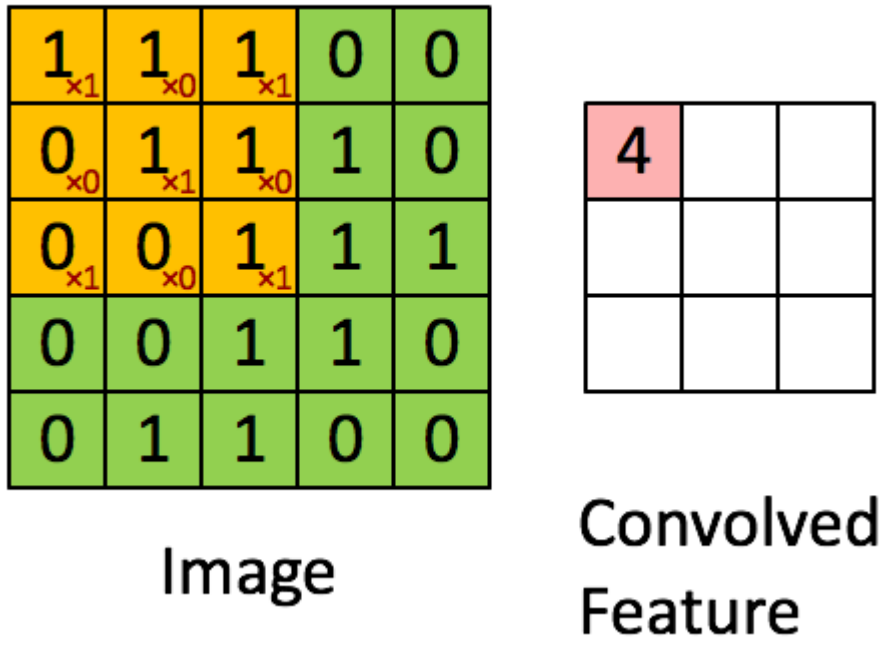


Source: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

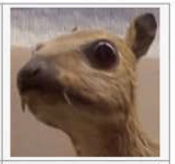
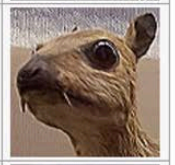
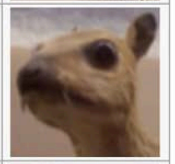
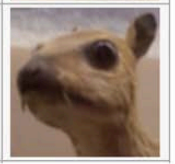
Convolution



Convolution operator: weighted sum where weights are from filter matrix

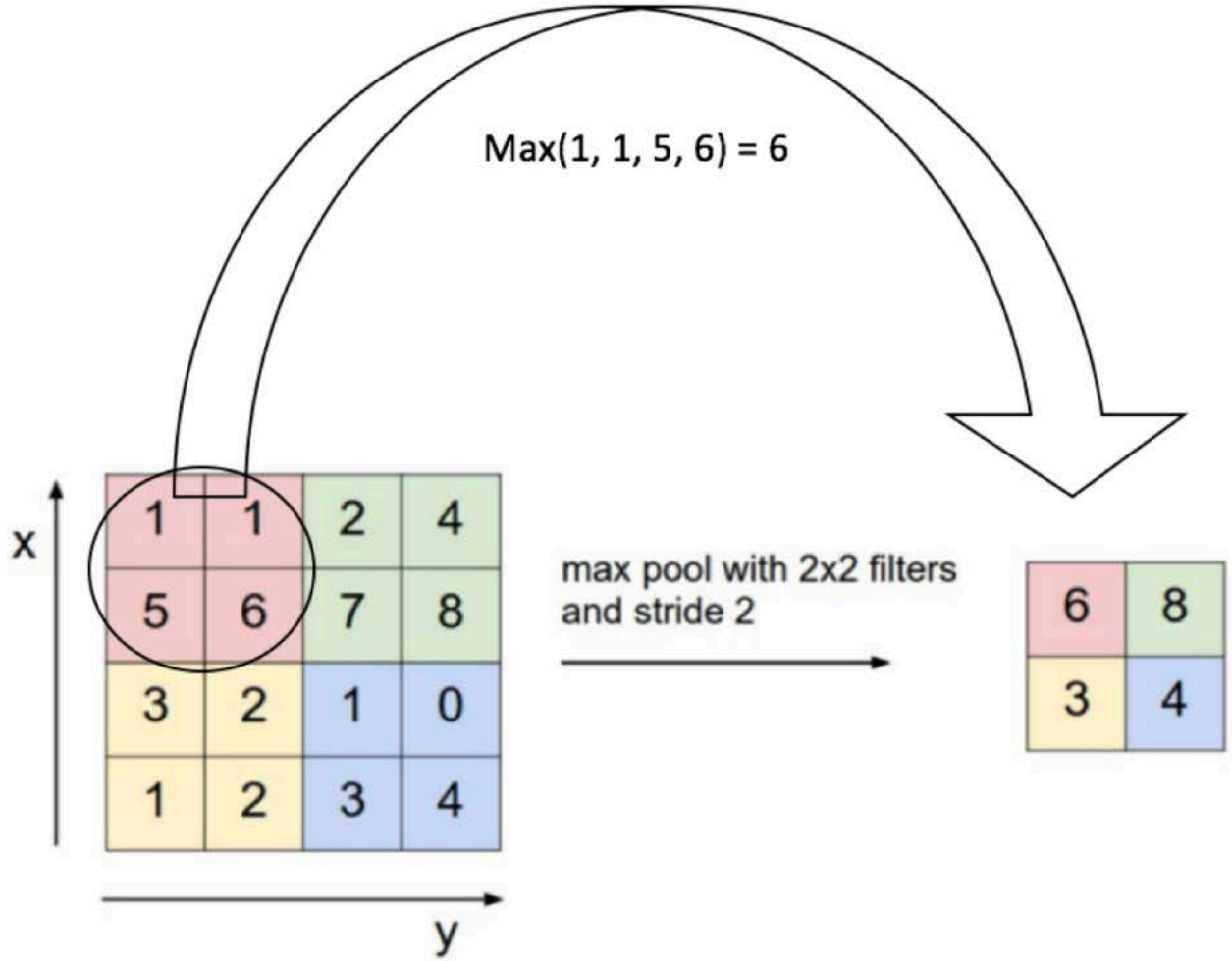


Effects of Different Filters

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)

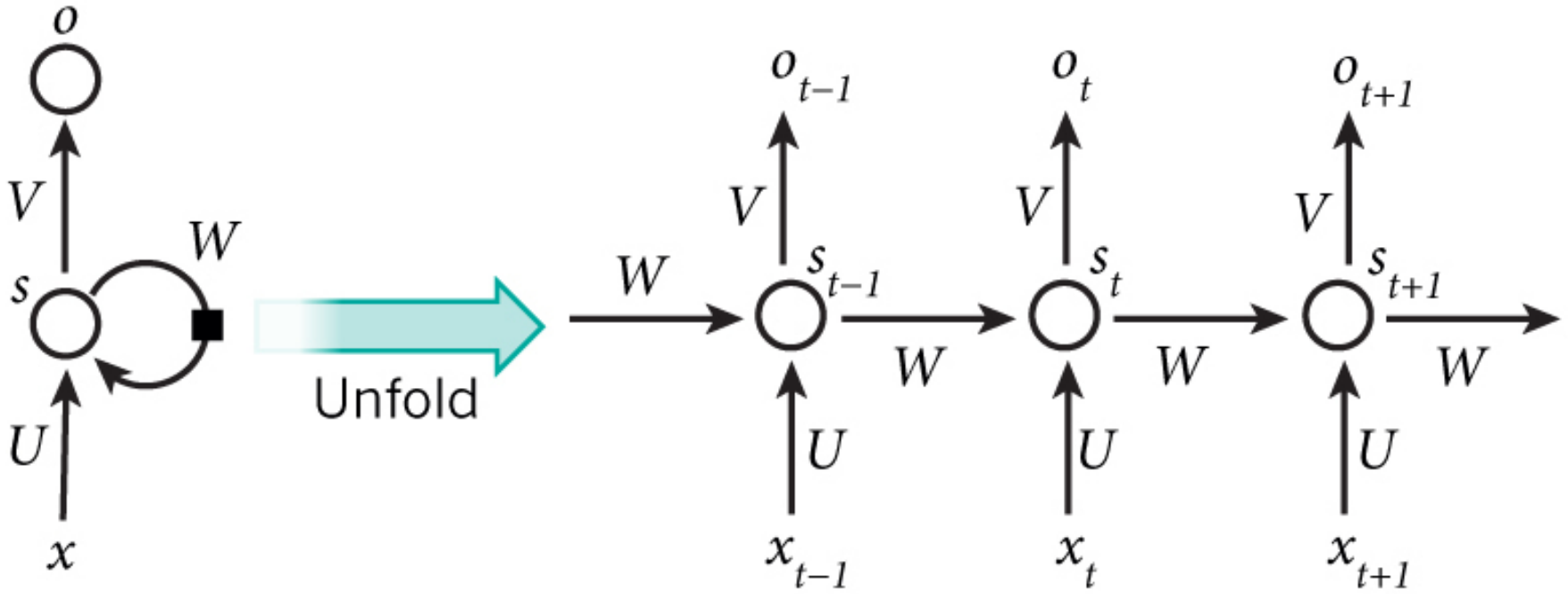
Pooling



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

Recurrent Neural Network

- Model sequential information

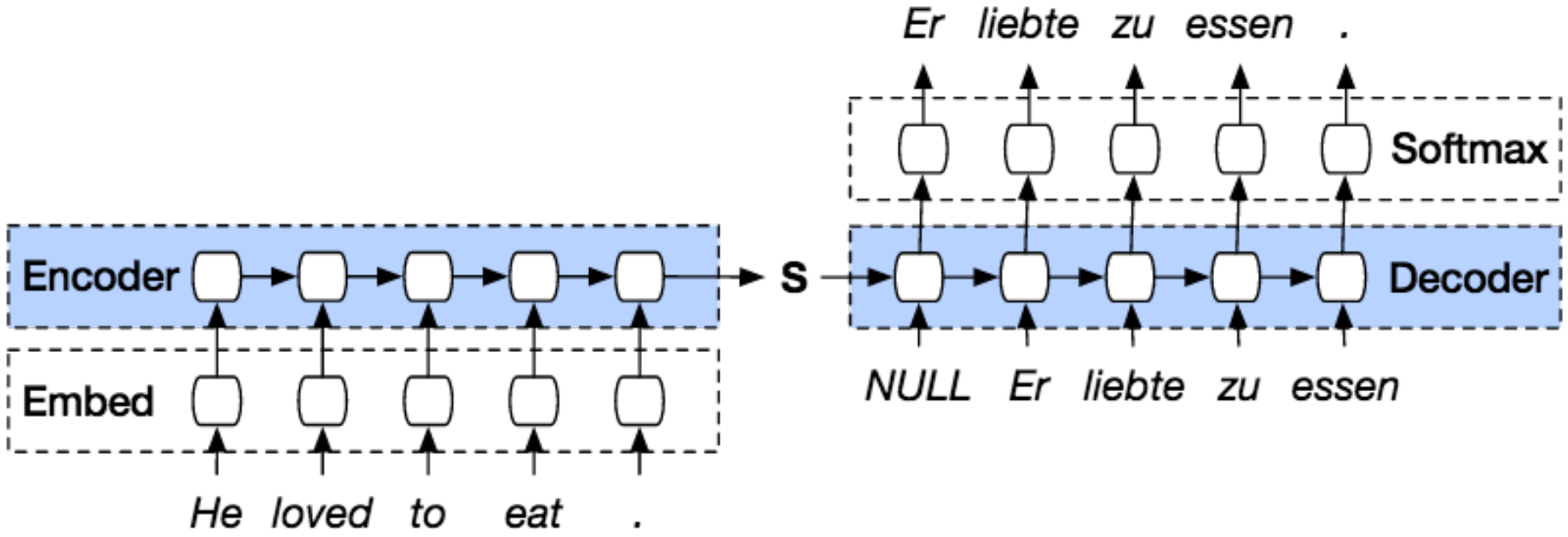


x_t: *input vector at time t*
s_t: *hidden state at time t*
o_t: *output vector at time t*

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

Application of Machine Translation

- Seq2Seq model



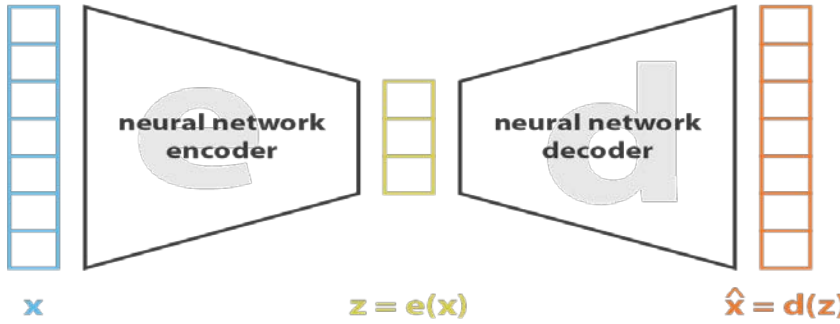
Variational Autoencoder

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

https://houxianxu.github.io/assets/project/dfcvae

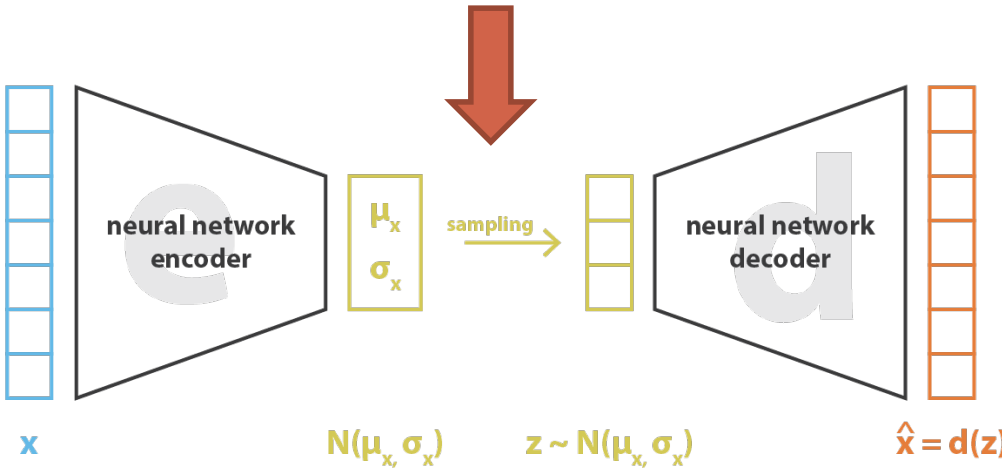
Variational Autoencoder

regular autoencoder



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

variational autoencoder



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

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

Other resources for variational autoencoder

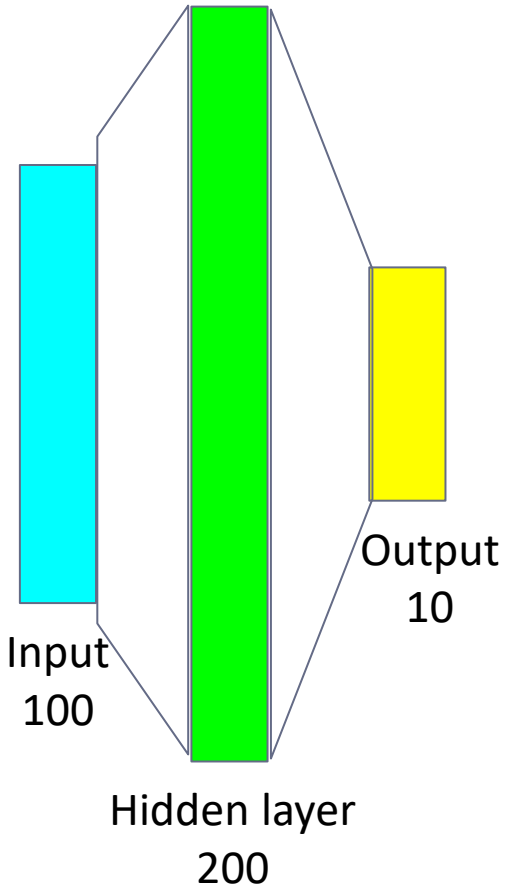
- https://www.youtube.com/watch?v=c27SHdQr4lw by Paul Hand, 2020
- https://arxiv.org/pdf/1906.02691.pdf, by Kingma and Welling, 2019 (An introduction to variational autoencoders)
- Stanford course: https://deepgenerativemodels.github.io/notes/vae/

Q&A

Implementation

- Define architecture
- Define loss
- Training
  - Define optimizer
  - Define training schedule
  - Hyper parameter tuning
  - …
- Inference

Build a MLP with Pytorch



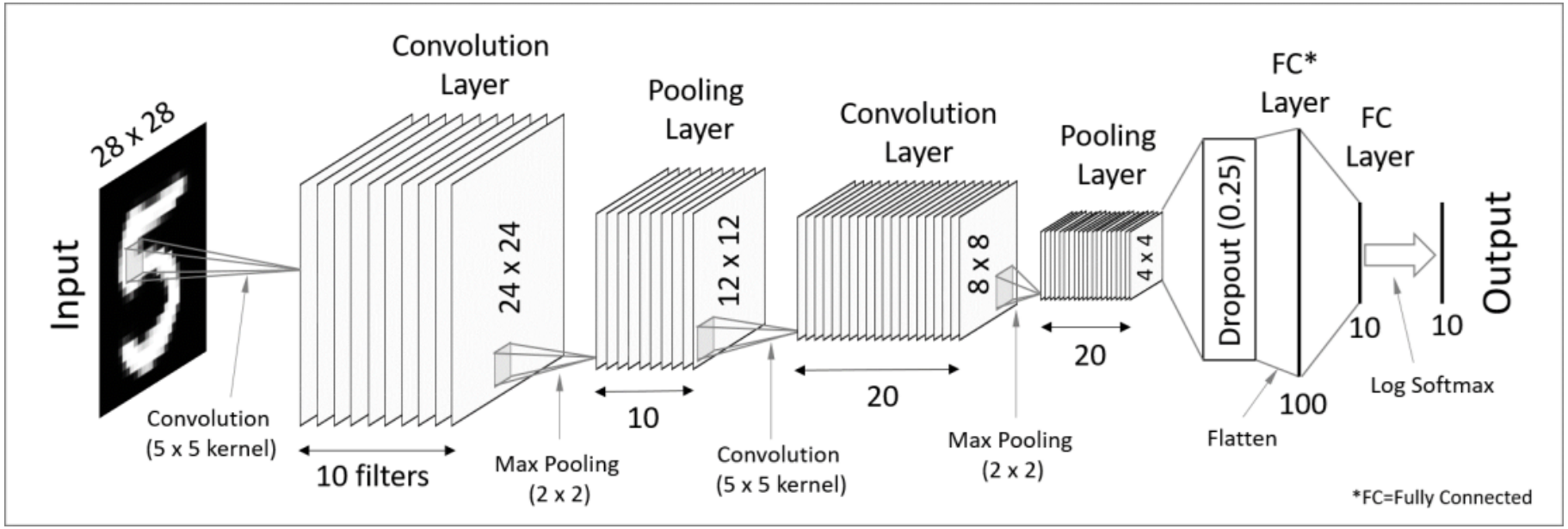

```python
import torch
import torch.nn.functional as F
class TinyModel(torch.nn.Module):
    def __init__(self):
        # run the __init__ function of its super class
        super(TinyModel, self).__init__()
        # define each layer of the NN
        # torch.nn.Linear(input_dim, output_dim)
        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

Q&A

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