

---

# **Neural Networks and Deep Learning**

**Dr. Jerome J. Braun**

## **This Lecture: Course Description and Logistics**

Course: Neural Networks and Deep Learning  
IE 7615  
Spring 2023

---

---

**Unauthorized distributing, redistributing, posting,  
and/or reposting, of the materials of this course  
(including course lectures, lecture slides, slide-sets,  
syllabi, guidelines, assignments, quizzes, exams,  
presentations, software, electronic media, etc.),  
is prohibited.**

# This Lecture

---



- Course description
- Course logistics

# Course Objectives

---

After successfully completing this course, you should:

- Have learned fundamental concepts of machine learning, neural networks and deep learning
- Have learned mathematical foundations of selected mainstream approaches and algorithmic paradigms in neural networks and deep learning
- Understand selected theoretical issues in neural networks and deep learning
- Understand the function, structure, and characteristics of neural networks
- Have learned to apply various neural networks to practical problems
- Be able to do hands-on work to design, implement, test, and use neural networks
- Have learned to design, implement, train, test, evaluate, and use neural networks

# Neural Networks and Deep Learning

## Course Topics (tentative)

Topic numbers are not necessarily the order of lectures

- |   |  |
|---|--|
| 1) Introduction and review of basic mathematical preliminaries.                                 | 13) Convolutional neural networks (CNNs) and CNN training.                 |
| 2) Artificial neural networks fundamentals.   | 14) CNN architectures, AlexNet, VGGNet, ResNet, Inception, Xception.       |
| 3) Probabilistic approaches, maximum likelihood principle, linear regression.                   | 15) Adaptive learning-rate algorithms, RMSProp, ADAM.                      |
| 4) Information theory basics for deep learning.   | 16) Deep learning in practice, practical advice for deep network training. |
| 5) Generative and discriminative approaches, logistic regression, linear discriminant analysis. | 17) Recurrent neural networks, LSTMs, GRUs, encoder-decoder architectures. |
| 6) Regularization, Bayesian models.   | 18) Autoencoders.  |
| 7) Biological neural networks and selected neuroscience basics.                                 | 19) Neural attention, transformers (Transformer neural networks), BERT     |
| 8) Multilayer perceptron networks.  | 20) Representation learning, unsupervised and semi-supervised learning.    |
| 9) Backpropagation and training of deep networks.   | 21) Deep generative models, generative adversarial networks (GANs).        |
| 10) Computational graphs for deep networks.   | 22) Probabilistic graphical models for deep learning.                      |
| 11) TensorFlow, Keras, PyTorch.   | 23) Deep-learning-related topics from statistical learning theory.         |
| 12) Regularization for deep learning.   | 24) Selected additional topics in deep learning.                           |

## Neural Networks and Deep Learning

### What We Will Study in This Course, and Why? (1 of 3)

- Modern AI (artificial intelligence) in reality is **neural networks and deep learning**
  - AI — currently a “street name” (media, popular press, advertising) for **neural networks and deep learning**
- Methods, techniques and algorithms of **neural networks and deep learning** are behind operation and successes of modern AI
- As such, **neural networks and deep learning** are indispensable to modern AI

## Neural Networks and Deep Learning

### What We Will Study in This Course, and Why? (2 of 3)

- Neural networks and deep learning skills— among top in-demand skills now and in foreseeable future
  - Big and growing demand for neural-networks-and-deep-learning professionals
    - Companies across industry and commerce need this expertise for their AI applications
    - Trend will continue as AI continues to pervade more aspects of human activity
  - Numerous opportunities in industry and in academia
- High potential for “transformative” and “disruptive” advances
- BUT key to your success in **neural networks and deep learning** is learning the **SCIENCE** aspects of this field !
  - Implementation skills are NOT enough — it is NOT about being a “toolkit driver” !
  - You need to KNOW underlying theory and math behind code and software libraries (toolkits)
    - And you need to be able to demonstrate it (e.g., in job interviews)

J. Braun

7

## Neural Networks and Deep Learning

### What We Will Study in This Course, and Why? (3 of 3)

- To succeed in this field, you need to know **theory and mathematics of neural networks and deep learning**
  - I will teach you these in this course
- In this course, you will also gain **practical hands-on experience**
  - Course project (referred to as “final project”)

**Learn mathematical and theoretical foundations**

- Learn science of neural networks

**Gain insights and intuitions****Learn techniques and algorithms****Learn the science behind modern deep-learning software-libraries, such as TensorFlow and Keras**

- To enable you to use these toolkits knowledgeably (wisely)

**Gain practical HANDS-ON experience**

- Course Project (referred to as “Final Project”)
- Learn to *implement* neural networks and deep-learning algorithms and techniques
- Learn to use modern popular software-libraries such as
  - TensorFlow
  - Keras
  - PyTorch

J. Braun

8

# This Lecture

---

- Course description
- · Course logistics

# Main Study Sources

---

- My lectures
  - Notes you take during class
  - Discussions in class
- Lecture slides for your study in this course will be placed on the Canvas system (“Canvas”)
- Required textbook (see next slide “Books”)
- Check Canvas often
  - Lecture slides
  - Syllabus
  - Homework assignments
  - Announcements (in addition to announcements made in class)
  - Other course-related items

# Books

Refer to course syllabus.

## Required textbook:

Goodfellow, I., Bengio, Y., Courville, A., '[Deep Learning](#)', MIT Press, 2016.

## Supplemental references:

Bishop, C. M., '[Pattern Recognition and Machine Learning](#)', Springer, 2006.

Hastie, T., Tibshirani, R. and Friedman, J., '[The Elements of Statistical Learning](#)', 2nd Ed., Springer. 2017.

Haykin, S., '[Neural Networks and Learning Machines](#)', 3rd Ed., Pearson, 2009.

Koller, D. and Friedman, N. '[Probabilistic Graphical Models](#)', MIT Press. 2009.

Murphy, K.P., '[Machine Learning: A Probabilistic Perspective](#)', MIT Press, 2012.

Duda, R.O., Hart, P.E., and Stork, D.G., '[Pattern Classification](#)', Wiley, 2001.

Scholkopf B. and Smola A., '[Learning with Kernels](#)', MIT Press, 2002.

Dayan, P. and Abbott, L.F., '[Theoretical Neuroscience: computational and mathematical modeling of neural systems](#)', MIT Press, 2001.

# Grading

Refer to course syllabus.

- Exams

- Midterm exam: 17% of total course-grade
- Final exam: 30% of total course-grade

If either midterm exam or final exam is not administered to *entire class*, respective percentage weight of non-administered exam will be distributed to remaining grade-components — however please note that if exam is administered to class, redistribution option will NOT be available to any individual student(s)

Participation in all administered exams is required

Non-participation or receiving a zero-score in either the midterm exam or in the final exam when administered, may result (regardless of any other grade-components, e.g., scores on other exams, final project, etc.) in failing the course

- Homework problem-sets — 8% total

- Self-graded, submit completed homework and grade, will be verified at random

- Final Project (40%)

- More on this in subsequent slide and in subsequent lectures

- Class participation (5%)

- Attendance, participation in discussions

# Homework — Problem-Sets

---

- **Problem-set may include mix of**
  - **Problems to solve (written)**
  - **Coding assignments**
    - ♦ **In Python**
  - **Homework — important part of your study, and helps you to prepare for project and exams**
- **Self-grading**
  - **You will submit your solutions (including code for coding assignments), and your own grade**
  - **We will randomly check to verify**

## Expectations, Policies, Student Academic Honesty and Integrity

---

Refer to course syllabus.

- **Comply with all Northeastern University policies, college/department policies, and rules of this course**
- **Academic honesty and integrity**
  - Cheating not tolerated — risk failing course and/or facing administrative actions
- **All work on exams must be student's own work**
- **Homework assignments must be student's own work**
  - Using past-years' solutions (from *any* source) is prohibited
  - Using websites to obtain solutions is prohibited
  - Students *may* engage in discussions of assignments with their peers currently enrolled in this course, and the TA or the instructor, provided that:
    - ♦ Following such interactions each student performs the assignment independently on his/her own (without referring to and/or copying notes from those discussions)
- **Refer to syllabus of this course and comply with all rules listed in that syllabus**

# Final Project

- Final Project is mandatory Refer to course syllabus.
- Students are expected to form self-organized teams of up to maximum of three students per team (team-members within each team collaborating to produce respective team's project)
- Mandatory project-related items include
  - Project proposal submission, project proposal presentation, proposal acceptance (proposals need to be approved by me), final paper submission, final presentation
    - All of these are mandatory no later than on their respective due-dates
    - These due-dates will be announced well ahead of time
  - Project-specific deliverables, in accordance with any particular accepted project-proposal
- Projects can be either application-oriented (most common) or theoretical
  - Datasets used must be from reputable public machine-learning research repositories, and must be available without restrictions
- Project proposals subject to acceptance and approval by me
  - Detailed *Project Guidelines* will be provided, and will be covered in class.
  - First part of Project Guidelines will describe project-proposal preparation in detail, and will also be covered in detail in class.

J. Braun

15

## Submitting Course-Deliverables into Canvas

- Course-deliverables will be submitted into Canvas ONLY
  - E.g., homework assignments, any and all final-project deliverables
- Any other form of submission is not allowed
  - Submissions by email are not allowed

J. Braun

16

# Course Prerequisites

- **Knowledge of fundamentals of**
  - Probability
  - Basic linear algebra (matrix algebra)
  - Calculus
- **At least rudimentary knowledge of Python programming language**
  - Knowledge of Matlab or other programming languages such as C/C++ could be an additional advantage

Slideset covering review of some of these will be provided

**Making up any deficiencies is, obviously, individual responsibility of each student**

**Questions?**

---

# **Neural Networks and Deep Learning**

**Dr. Jerome J. Braun**

## **This Lecture: Introduction — Part 1**

Course: Neural Networks and Deep Learning  
IE 7615

---

---

**Unauthorized distributing, redistributing, posting,  
and/or reposting, of the materials of this course  
(including course lectures, lecture slides, slide-sets,  
syllabi, guidelines, assignments, quizzes, exams,  
presentations, software, electronic media, etc.),  
is prohibited.**

# This Lecture

---

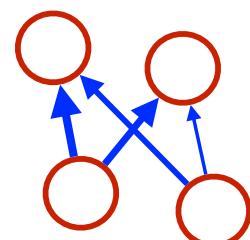


- Neural networks and deep learning — field
- Applications

## What is Artificial Neural Network (ANN)?

---

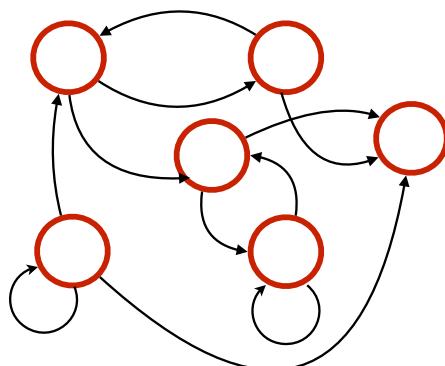
- **Units** with associated function
  - Also referred to in literature as nodes or “neurons”
- **Connections** — weighted **links** between units
- **Learning algorithm**
  - Typically — modify weights of connections
  - Sometimes — modify units, network topology
- **Conventional artificial neural networks (ANNs) are NOT models of biological neural networks or brain !**
  - We will see this later in this course



# ANN Topologies

---

- Most general — fully/arbitrarily connected



## List of *Some* ANN Architecture Categories

---

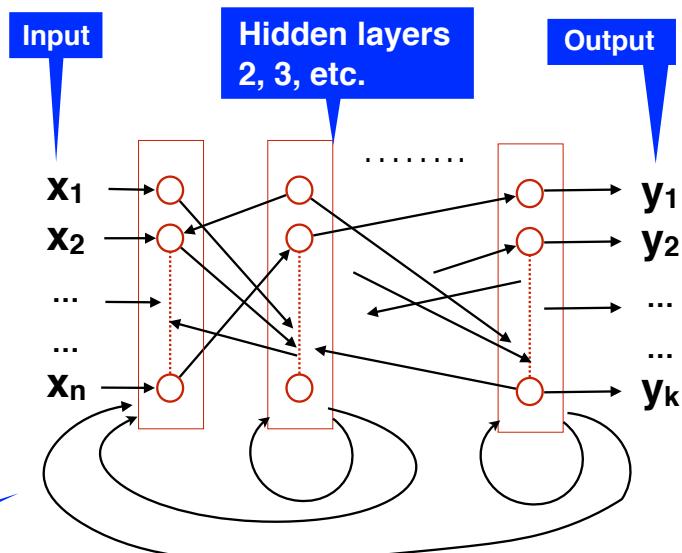
- “Shallow” multilayer perceptron (MLP) networks
- Deep MLP networks
- Convolutional networks
- Recurrent networks, LSTMs, GRUs, ...
- Autoencoders
- Transformers, BERT, ...
- Generative adversarial networks (GANs)
- ... and other

We will study them  
in this course

# ANN Topologies – Layered

- **Layered**
- **Any number of layers**
- **Retrograde connections**
  - Within layers
  - To previous layers

Retrograde  
connections

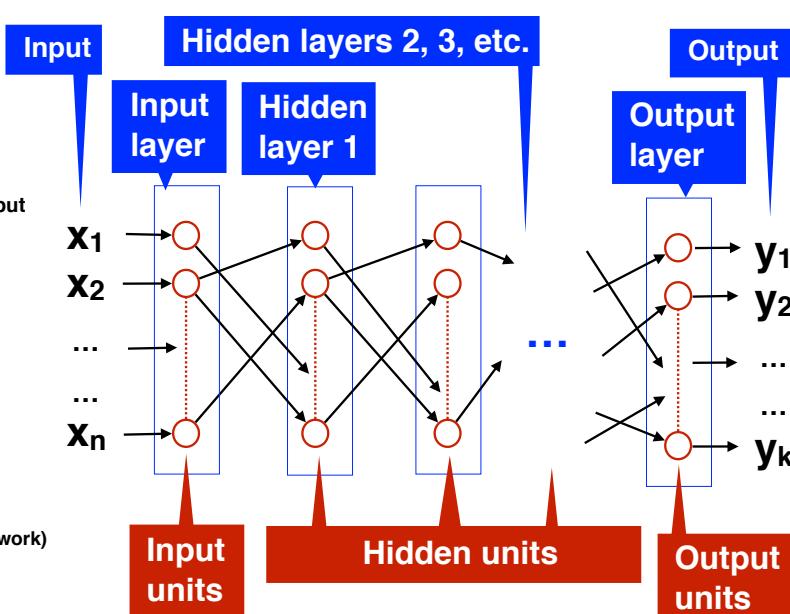


J. Braun

7

# ANN Topologies – Feedforward

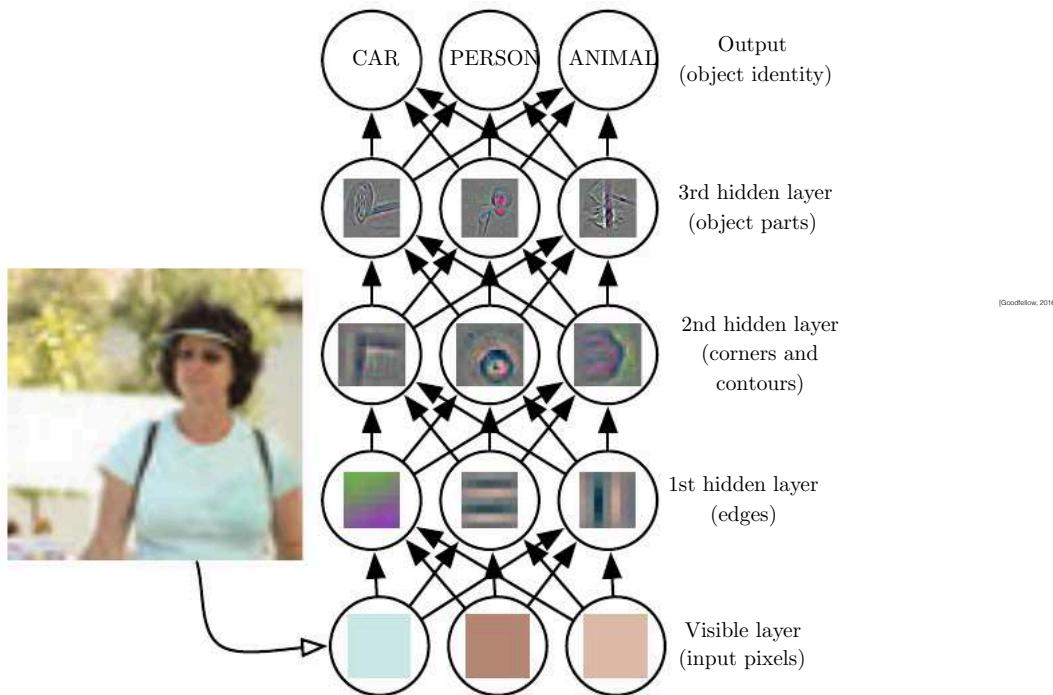
- Layered
- Only forward connections
  - Direction from input to output
  - From layer  $i$  to layer  $i + 1$
- Any number of layers
- Layers
  - Input layer
  - Hidden layers
    - One (shallow network)
    - More than one (deep network)
  - Output layer



J. Braun

8

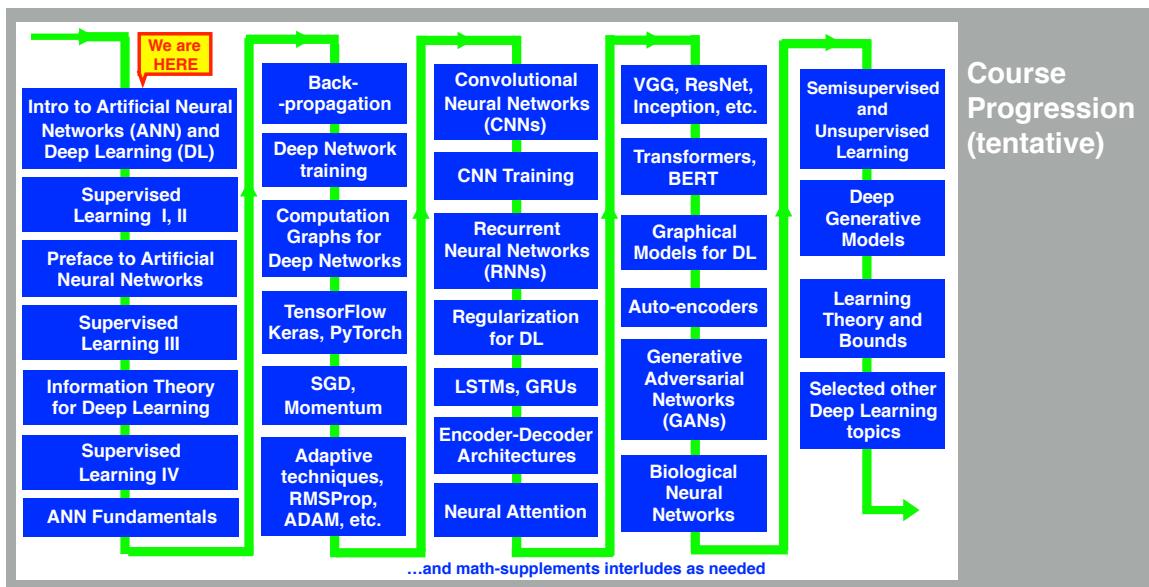
# Deep Learning Model



9

# Course Progression

- Selected machine-learning fundamentals are necessary for understanding neural networks and deep learning — we will study these necessary fundamentals in the first part of the course
- Then we will focus on neural networks (ANN) and deep learning (DL) in detail



# Neural Networks and Deep Learning

Machine Learning

Neural Networks

Deep Learning

Here "Neural Networks"  
means "*Artificial* Neural Networks"

Neural networks have evolved  
into dominant, large and  
uniquely powerful domain

Solve problems too difficult for  
any other known approaches

Neural Networks

Deep Learning

**Deep Learning = deep neural networks !**

J. Braun

11

## Machine Learning

- Many tasks not feasible to “pre-program”
  - Not feasible to find “rules”
  - Too much variability
  - Examples: handwriting recognition (e.g., zipcode on envelopes), speech recognition
- Machine learning — algorithmic methods to learn
  - From data, from experience
  - Arthur Samuel (1959): *Machine Learning — field of study that gives computers ability to learn without being explicitly programmed*
- Learning from data
  - Some data
  - Lots of data (“Big Data”)
  - Little data

J. Braun

12

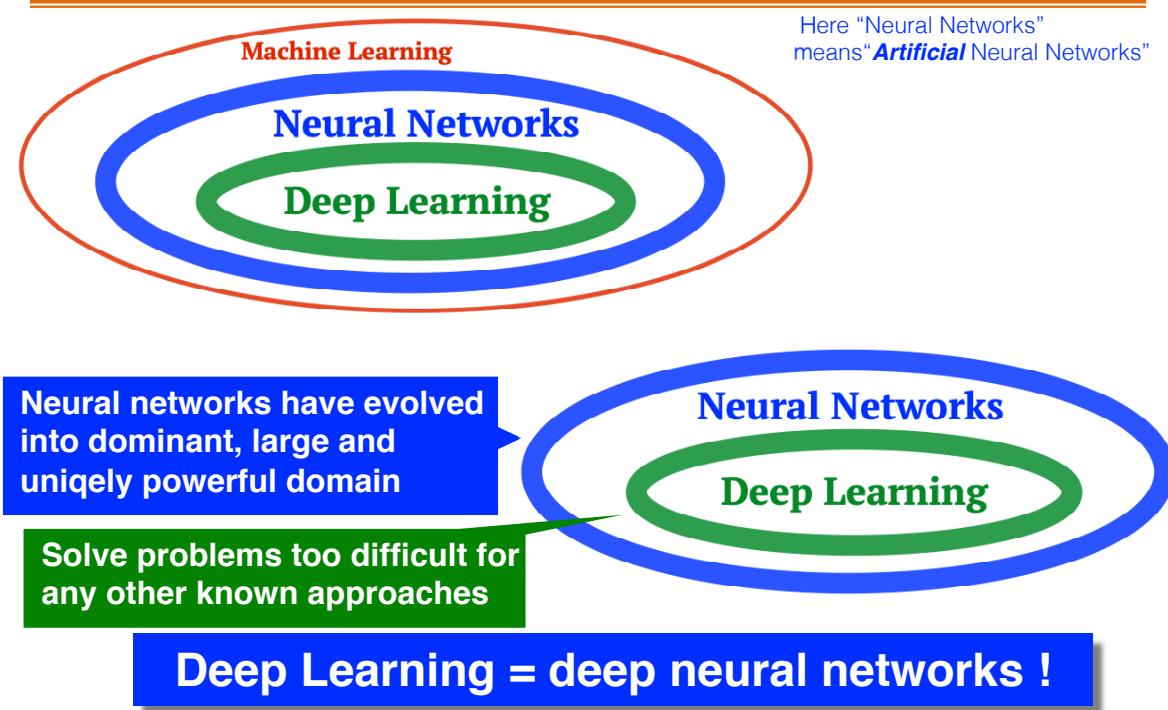
## Learning From Data – Algorithmic Paradigms

- Classical statistical (regression, decision-trees, etc.)
- Clustering (nearest-neighbor, etc.)
- Component analyses (PCA, ICA, DCA, etc.)
- Hidden Markov models (HMM)
- Support vector machines (SVM)
- ... etc. ...
- Artificial neural networks and deep learning
  - Feedforward neural networks
  - Recurrent neural networks
  - ... etc. ...

J. Braun

13

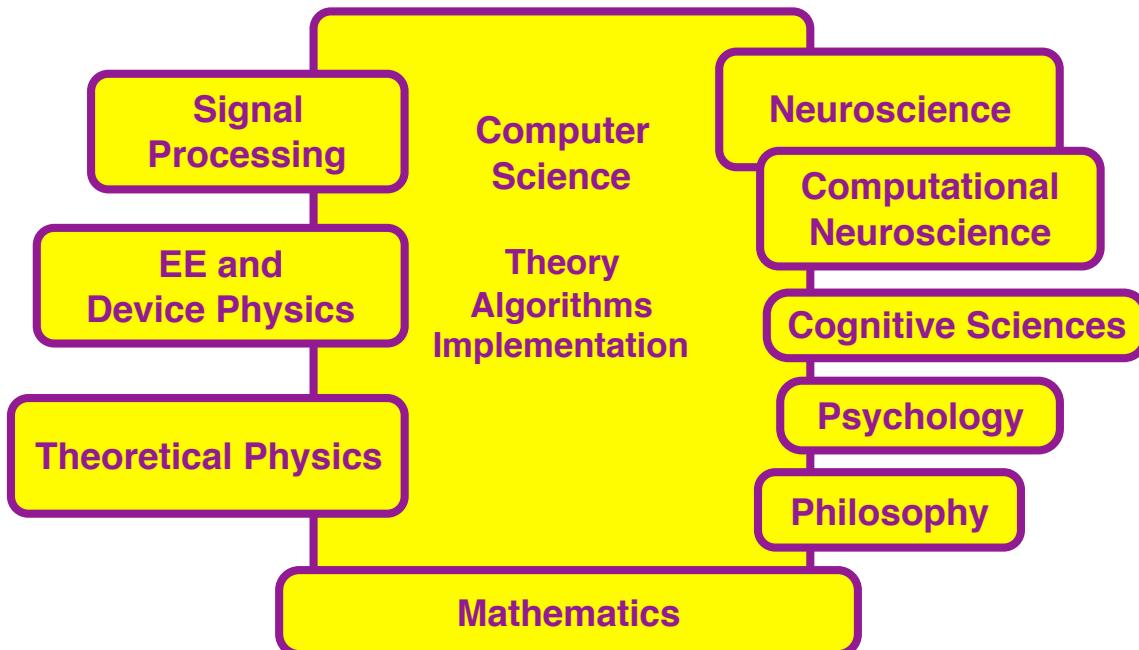
## Neural Networks and Deep Learning (1)



J. Braun

14

## Neural Networks and Deep Learning (2)



J. Braun

15

## Neural Networks and Deep Learning (3)

- **Learning in biological species vs. machines**
  - Learning vs. innate capabilities in biological species
- **Reasoning vs. learning**
  - Machine Reasoning
  - Learning to reason
- **Deep Learning and Artificial Intelligence (AI)**
  - At present neural networks and deep learning are necessary for AI



J. Braun

16

## Neural Networks and Deep Learning (4)

---

- In biological systems, robust learning is key to intelligent behaviors
  - Same for artificial systems!
- Robust learning is key for developing intelligent machines
- Central core of contemporary Artificial Intelligence (AI)
  - Modern AI is dominated by deep neural networks (deep learning)
- AI is currently “street name” for deep neural networks and deep learning
  - In media (TV, etc.), advertising, popular press

## Neural Networks and Deep Learning (5)

---

- Who cares? Everybody!
  - Deep learning — currently “hottest” area!
- Companies racing to make/use systems based on neural networks and deep-learning
  - Amazon, Google, Facebook, Apple, Microsoft, and other high-tech giants
  - Scores/hundreds of other companies in industry and commerce
  - From healthcare, through manufacturing, to entertainment — and many more!
  - Wide range of areas and applications

# This Lecture

---



- Neural networks and deep learning — field
- Applications

## Neural Network and Deep Learning — Applications

---

A few examples (out of many)	
Machine Vision	Speech & Language Processing (NLP)
Face Recognition, Object Recognition	Autonomous Vehicles
Self-driving cars	Robotics
Forecasting	Assistive Robotics
Weather, climate, etc.	Bioinformatics
Computational biology, neuroscience	“X-”informatics
Analytics (text, video, etc.)	Economics, Financial forecasting
Medical diagnostics	Insurance
Biomed data analysis; healthcare	Fraud detection
Drug development	Image/photo tagging
Personalized medicine	
... many others...	

- Span of potential applications enormous
- But beware of misapplication and overkills!

# Application Examples

## Handwriting Recognition

5 7 8 a  
2 9 b

## Face Recognition

In 2011,  
IBM Watson system  
won “Jeopardy!” TV game  
against human champions

Movie suggestions, e.g.,  
“Other Movies You Might Enjoy”

J. Braun

21

# Self-Driving Cars

- Google:



- Others

J. Braun

22

# Speech Processing

- Speech Recognition
- Language Identification
- Speaker Recognition / Verification
- Accent Recognition
- Speech Synthesis
- ...more

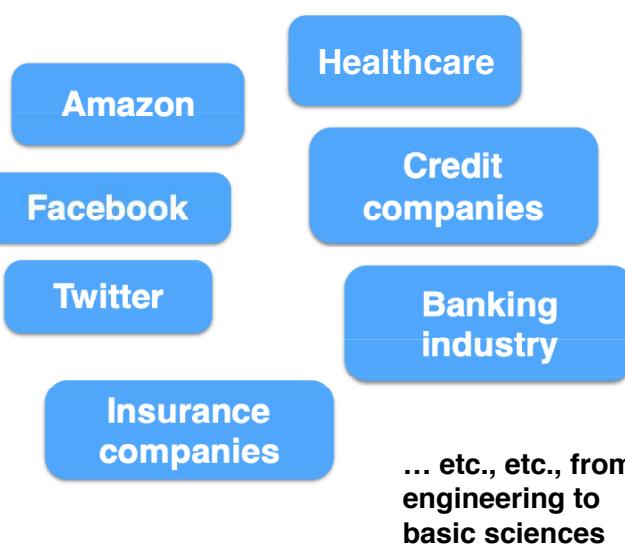
Siri

J. Braun

23

## “Big Data” and Deep Learning

Huge and growing data  
collected, stored and available



Deep neural networks –  
deep-learning systems

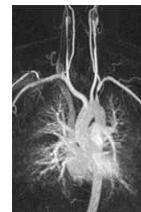
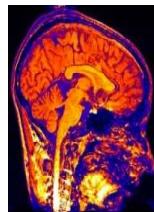
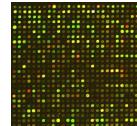
Uniquely suitable for  
exploiting very large  
amounts of data  
("big data")

J. Braun

24

# Biomedical Applications

- Large and growing range of applications in medicine, healthcare, and other life-sciences areas
  - High potential for revolutionizing disease diagnostics and treatment
- Examples
  - Bioinformatics and biomedical data analysis
    - E.g., gene expression analyses
  - Medical imagery analysis for disease diagnostics
    - E.g., abnormality detection, lesion classification (diagnosis)

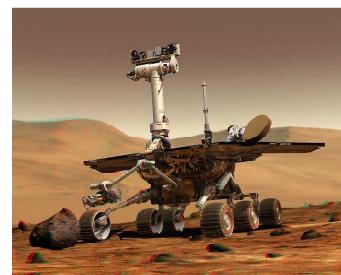


25

# Robotics Applications



Robocup (Robocup.org)



NASA Mars Spirit Rover (Wikipedia.org)

- Wide range of applications
- Examples: assistive robotics, medical robotics, disaster robotics, manufacturing

# Deep Learning for Medical Diagnostics

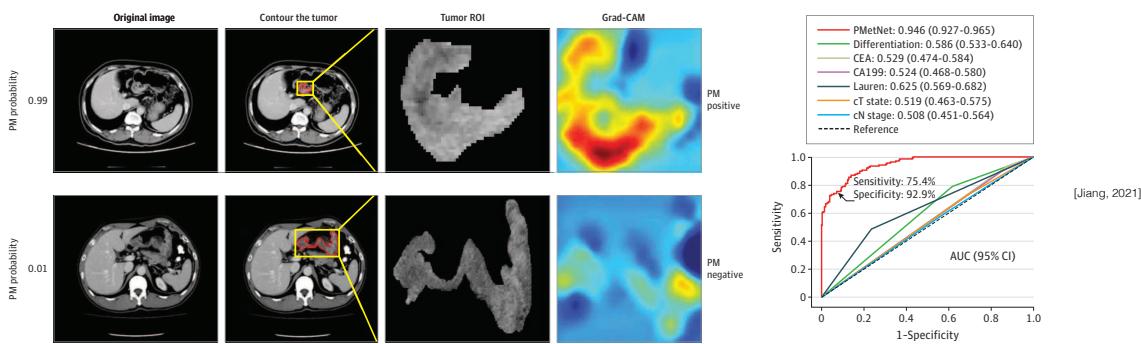
- In popular press referred to as AI
- In reality — deep artificial neural networks (deep learning)
  - Convolutional networks, etc.
- Wide range of diagnostic applications and successful results
  - Lung disease, including malignancies
  - Skin lesions and malignancies
  - Breast cancer
  - and other...
- Arguably revolution of medicine
  - Current results indicate performance superior to human experts

J. Braun

27

## Deep Neural Networks for Noninvasive Prediction of Occult Peritoneal Metastasis in Gastric Cancer

- Jiang *et al.*, JAMA Network Open, January 5, 2021
- Noninvasive preoperative (pre-surgery) assessment of occult peritoneal metastasis of gastric cancer
- Potentially useful to avoid unnecessary surgery and risk of associated complications
- CT imagery
- 1978 patients
- Densely connected convolutional neural network (CNN)
- Discrimination performance of network substantially higher than conventional clinicopathological factors

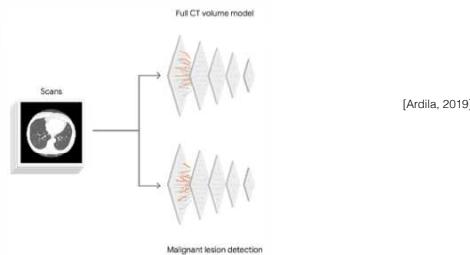


J. Braun

28

# Deep Neural Networks for Lung Cancer Screening

- Ardila et al., Nature Medicine, May 2019
- Predict lung cancer risk by comparing patient's current and prior CT imaging
- Deep convolutional neural networks (CNN)
- 6,716 National Lung Cancer Screening Trial (NLST) cases
- 94.4% AUC performance
- Performance better or comparable with human readers (radiologists)

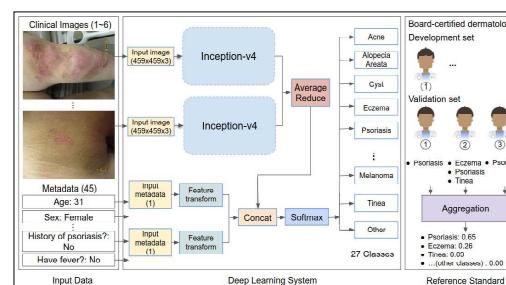


J. Braun

29

# Deep Neural Networks for Skin Lesion Classification

- Liu et al., 2019
- Differential diagnosis of 26 skin conditions from photographs and medical histories
- 14,021 development cases, 3,756 evaluation cases
- Variable number of deep convolutional neural network modules to process images
  - Inception-v4
- Shallow module for patient demographic information and medical history (metadata)



[Liu, 2019]

J. Braun

30

# Deep Neural Networks for Breast Cancer Screening

- McKinney *et al.*, Nature, January 1, 2020
- Breast cancer prediction from mammograms
- Ensemble of three deep-learning models
- Exploit ImageNet, RetinaNet, ResNet
- All models implemented in TensorFlow
- Platform includes Google TPU hardware
- System performance — potential to perform better than trained radiologists
  - False positives reduction: 5.7% and 1.2% (US and UK)
  - False negative reduction: 9.4% and 2.7%
  - Able to generalize from UK data to US data
  - Able to outperform human experts
    - ♦ Six readers (radiologists)
    - ♦ AUC-ROC performance higher than that of human readers by 11.5% margin

J. Braun

31

# Deep Neural Networks for Cardiovascular Risk Prediction

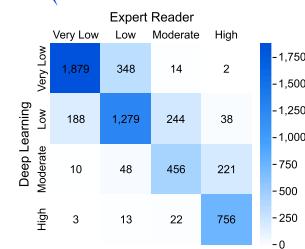
- Zeleznik *et al.*, Nature Communications, Jan. 2021
- Coronary artery calcium — predictor of cardiovascular events
- Visible on all CT chest scans computed tomography (CT) scans
  - But quantification requires expertise, time, and specialized equipment
- Robust automatic quantification by deep-learning system
  - Convolutional neural networks
- 20,084 individuals from asymptomatic, and stable and acute chest pain cohorts
- High correlation of deep-learning system with quantification by expert readers
  - And robust test-retest reliability

Example patients' results



Heart contour (blue)

Coronary calcium (orange)



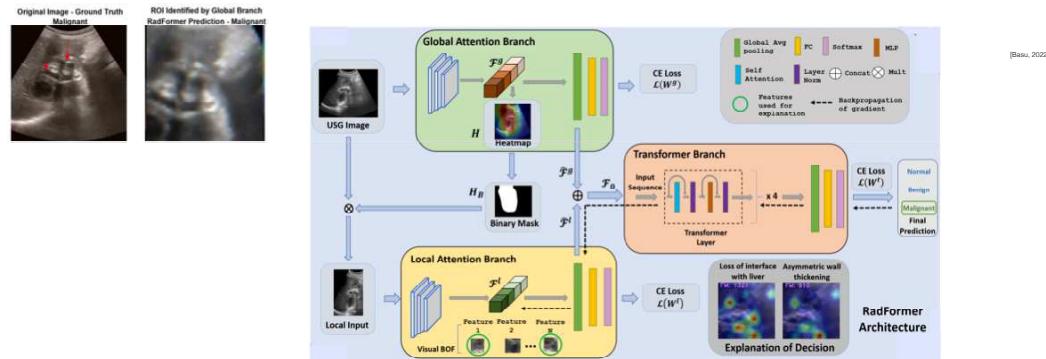
[Zeleznik, 2021]

J. Braun

32

# Deep Neural Networks for Gallbladder Cancer Diagnostics

- Basu *et al.*, 2022
- Diagnostics of gallbladder malignancies
- Input: ultrasound sonography images
- Transformer network architecture
- Basu *et al.* compared system results with conclusions of two expert radiologists
  - Found system performance was better

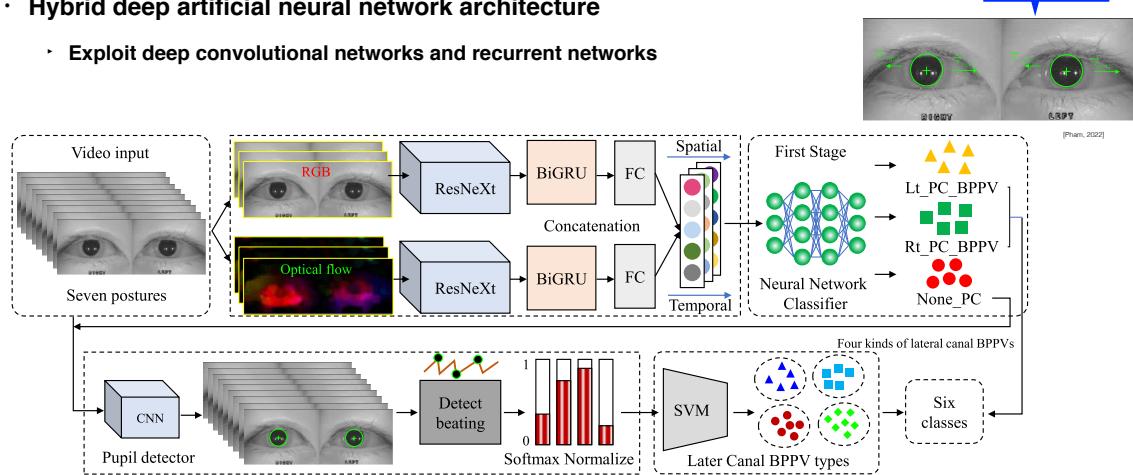


J. Braun

33

# Deep Neural Networks for BPPV Diagnosis

- Pham *et al.*, Oct. 2022
- Diagnostics of benign paroxysmal positional vertigo (BPPV) types
  - Posterior canal types (Left, right), Lateral canal types: geotropic BPPV (left, right), apogeotropic (left, right)
- Input: video stream of patient eye-motion during diagnostic medical exam (Dix-Hallpike test)
- Hybrid deep artificial neural network architecture
  - Exploit deep convolutional networks and recurrent networks

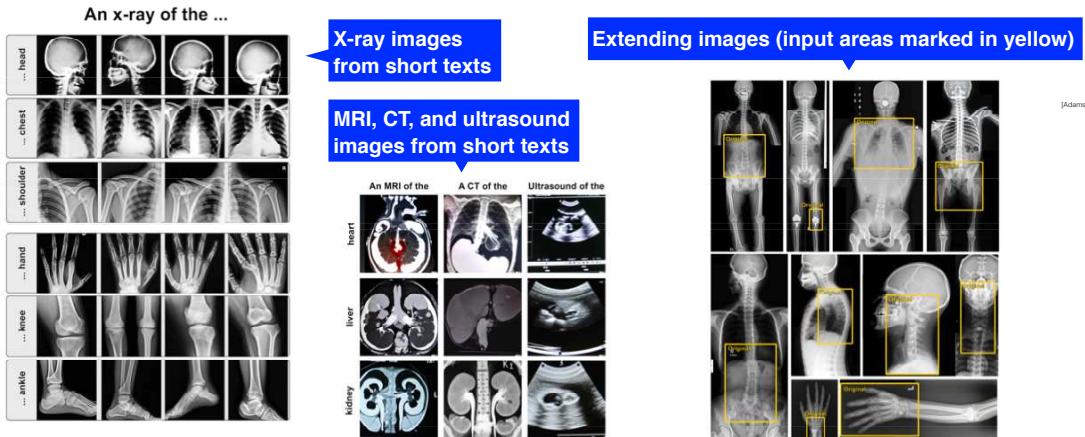


J. Braun

34

# Deep Generative Models for Medical Imagery

- Adams et al., 2022
- AI in radiology
  - Using DALL-E 2 generative model for text-to-image generation, image augmentation, and manipulation
  - DALL-E 2 learns relevant representations of X-ray images
    - + Zero-shot text-to-image generation of new images, continuation of image beyond original boundaries

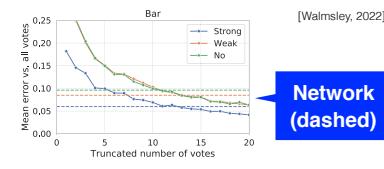
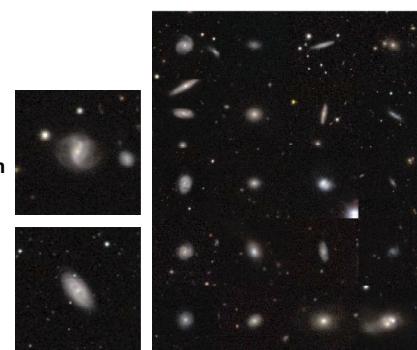


J. Braun

35

# Deep Neural Networks for Understanding Universe

- Walmsley et al., MNRAS 509, 2022
  - Deep neural networks for astronomy and astrophysics research
  - Visual morphological classification of galaxies from images
    - + Morphology of galaxies— key to understanding galactic evolution
- Data — *Dark Energy Camera Legacy Survey* images of galaxies
- Ensemble of convolutional neural networks
  - Exploit EfficientNet-B0 architecture with modifications
- Predict morphology features of galaxies
  - E.g., spiral arms, bars, etc.
  - Measured against confident volunteer classifications
    - + Galaxy Zoo volunteers
  - Trained networks reach up to 99% accuracy
    - + Measured against ~10 volunteers, could be viewed as achieving superhuman performance



J. Braun

36

# **Questions?**

---

# **Neural Networks and Deep Learning**

**Dr. Jerome J. Braun**

## **This Lecture: Math Preliminaries (Review)**

Course: Neural Networks and Deep Learning  
IE 7615

---

---

**Unauthorized distributing, redistributing, posting,  
and/or reposting, of the materials of this course  
(including course lectures, lecture slides, slide-sets,  
syllabi, guidelines, assignments, quizzes, exams,  
presentations, software, electronic media, etc.),  
is prohibited.**

# This Lecture

---

Review selected basics of



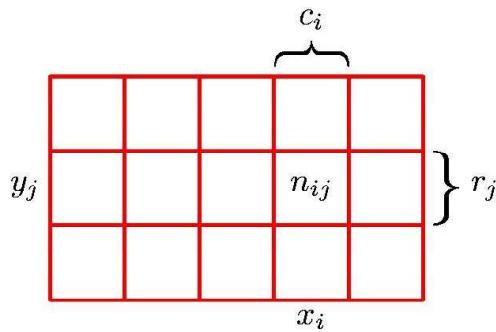
- Probability theory
- Linear algebra

## Probability Mass Function

---

- Domain of P — set of all possible states of  $x$
- Range  $\forall x \in \mathbf{x}, 0 \leq P(x) \leq 1$ 
  - ◆ Impossible event:  $P=0$
  - ◆ Guaranteed event:  $P=1$
- Normalization  $\sum_{x \in \mathbf{x}} P(x) = 1$

# Probabilities: Joint, Marginal, Conditional



[Bishop, Ch. 1.2]

Marginal probability

$$p(X = x_i) = \frac{c_i}{N}$$

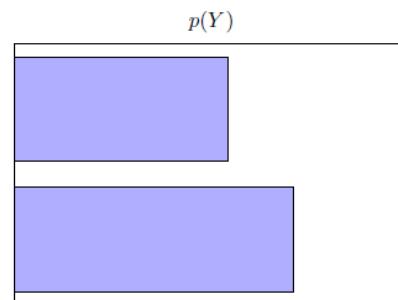
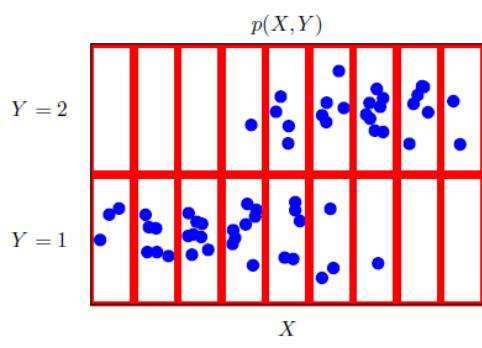
Joint probability

$$p(X = x_i, Y = y_j) = \frac{n_{ij}}{N}$$

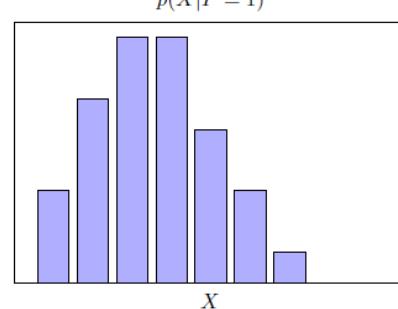
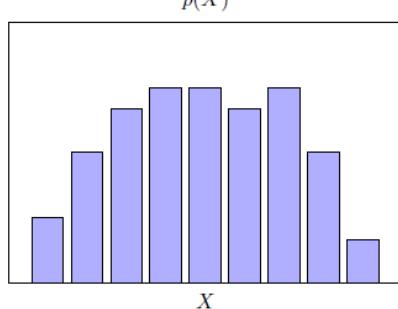
Conditional probability

$$p(Y = y_j | X = x_i) = \frac{n_{ij}}{c_i}$$

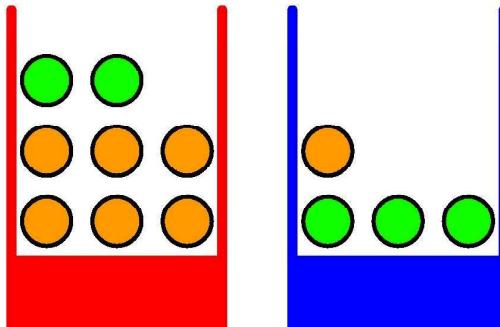
## Example



[Bishop]



# Basic Probability – Example

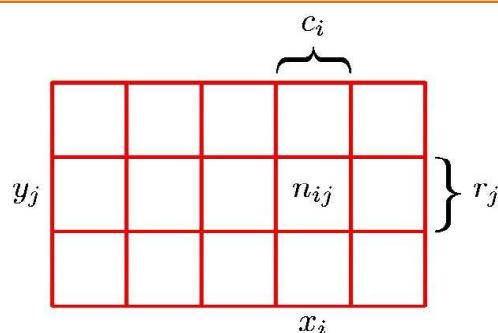


- **Experiment**
  - Randomly pick box (red or blue)
  - Then from that box randomly pick item (fruit — apple or orange)
  - Observe result, then replace item back
  - Repeat many times
- **Question: What is overall probability of picking apple?**

J. Braun

7

# Sum Rule



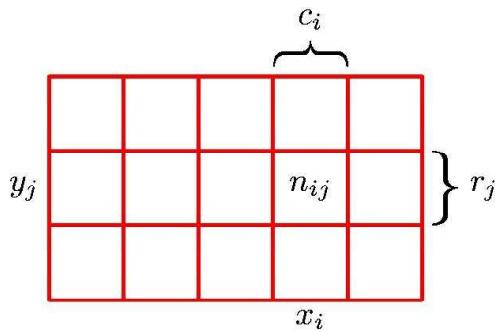
$$p(X = x_i) = \frac{c_i}{N} = \frac{1}{N} \sum_{j=1}^L n_{ij}$$

$$= \sum_{j=1}^L p(X = x_i, Y = y_j)$$

J. Braun

8

## Product Rule



[Bishop, Ch. 1.2]

$$\begin{aligned} p(X = x_i, Y = y_j) &= \frac{n_{ij}}{N} = \frac{n_{ij}}{c_i} \cdot \frac{c_i}{N} \\ &= p(Y = y_j | X = x_i) p(X = x_i) \end{aligned}$$

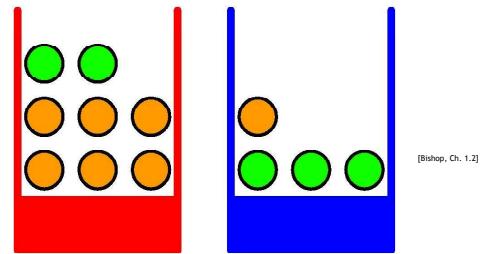
## Rules of Probability

- **Sum rule**  $p(X) = \sum_Y p(X, Y)$
- **Product rule**  $p(X, Y) = p(Y|X)p(X)$

# Basic Probability— Example

- Pick box at random
  - Turns out  $B=b$  (blue)
- What is probability of picking apple?
  - How about probability of picking orange?

$$\begin{aligned} p(F = a|B = r) &= 1/4 \\ p(F = o|B = r) &= 3/4 \\ p(F = a|B = b) &= 3/4 \\ p(F = o|B = b) &= 1/4. \end{aligned}$$



$$\begin{aligned} p(B = r) &= 4/10 \\ p(B = b) &= 6/10 \end{aligned}$$

Normalization

Apply sum and product rules

$$\begin{aligned} p(F = a|B = r) + p(F = o|B = r) &= 1 \\ p(F = a|B = b) + p(F = o|B = b) &= 1 \end{aligned}$$

$$p(F = a) = p(F = a|B = r)p(B = r) + p(F = a|B = b)p(B = b) = \frac{1}{4} \times \frac{4}{10} + \frac{3}{4} \times \frac{6}{10} = \frac{11}{20}$$

By sum rule

$$p(F = o) = 1 - 11/20 = 9/20.$$

# Bayes' Theorem

Likelihood

Prior

$$p(Y|X) = \frac{p(X|Y)p(Y)}{p(X)}$$

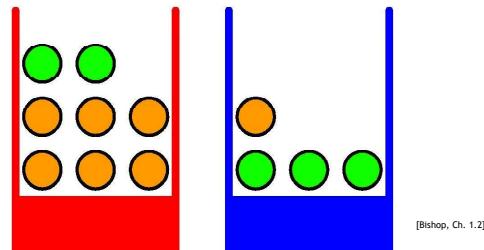
$$p(X) = \sum_Y p(X|Y)p(Y)$$

Posterior  $\propto$  Likelihood x Prior

# Using Bayes' Theorem – Example

- Told that fruit picked was orange
- From which box did it come?
  - How about probability of picking orange?

Reversing conditional probability and using Bayes' Theorem



$$p(B = r) = 4/10$$
$$p(B = b) = 6/10$$

$$p(B = r|F = o) = \frac{p(F = o|B = r)p(B = r)}{p(F = o)} = \frac{3}{4} \times \frac{4}{10} \times \frac{20}{9} = \frac{2}{3}$$

J. Braun

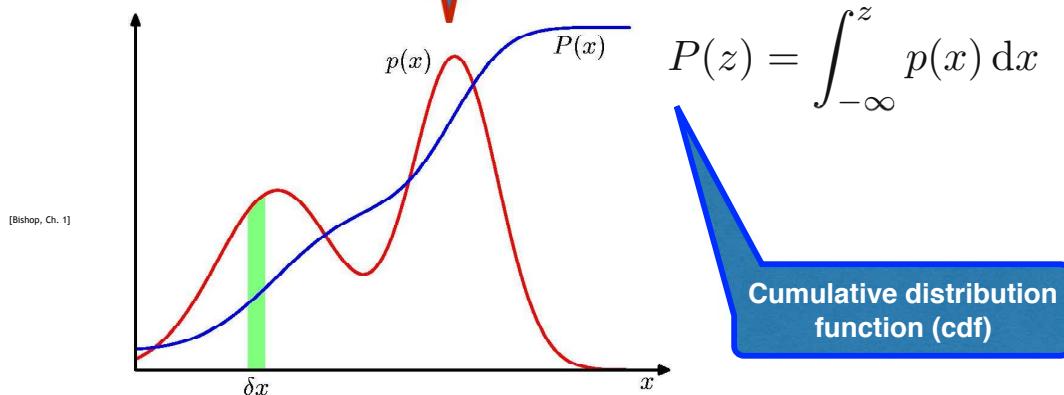
13

# Probability Densities

$$p(x) \geq 0 \quad p(x \in (a, b)) = \int_a^b p(x) dx$$

Probability density function (pdf)

$$\int_{-\infty}^{\infty} p(x) dx = 1$$



J. Braun

14

# Multivariate Probability Densities

Continuous variables

$x_1, \dots, x_D$

Joint probability density

$p(\mathbf{x}) = p(x_1, \dots, x_D)$

Infinitesimal volume

$p(\mathbf{x})\delta\mathbf{x}$

$p(\mathbf{x})$  must satisfy

$$\begin{cases} p(\mathbf{x}) \geq 0 \\ \int p(\mathbf{x}) d\mathbf{x} = 1 \end{cases}$$

over entire D-dim space

## Continuous Densities

- For continuous variables
- Sum and product rules apply also in case of continuous densities
- For instance if  $x$  and  $y$  are real variables:

$$p(x) = \int p(x, y) dy$$

$$p(x, y) = p(y|x)p(x)$$

- Bayes' theorem also applies

# Expectations

Discrete

$$\mathbb{E}[f] = \sum_x p(x)f(x)$$

Continuous

$$\mathbb{E}[f] = \int p(x)f(x) dx$$

Conditional (discrete)

$$\mathbb{E}_x[f|y] = \sum_x p(x|y)f(x)$$

Approximate

$$\mathbb{E}[f] \simeq \frac{1}{N} \sum_{n=1}^N f(x_n)$$

Linearity of expectations

$$\mathbb{E}_X[\alpha f(x) + \beta g(x)] = \alpha \mathbb{E}_X f(x) + \beta \mathbb{E}_X g(x)$$

# Variance and Covariance

Variance

$$\text{var}[f] = \mathbb{E} \left[ (f(x) - \mathbb{E}[f(x)])^2 \right] = \mathbb{E}[f(x)^2] - \mathbb{E}[f(x)]^2$$

Covariance

$$\text{Cov}(f(x), g(y)) = \mathbb{E} [(f(x) - \mathbb{E}[f(x)]) (g(y) - \mathbb{E}[g(y)])]$$

Covariance  
(scalar  $x, y$ )

$$\begin{aligned} \text{cov}[x, y] &= \mathbb{E}_{x,y} [\{x - \mathbb{E}[x]\} \{y - \mathbb{E}[y]\}] \\ &= \mathbb{E}_{x,y}[xy] - \mathbb{E}[x]\mathbb{E}[y] \end{aligned}$$

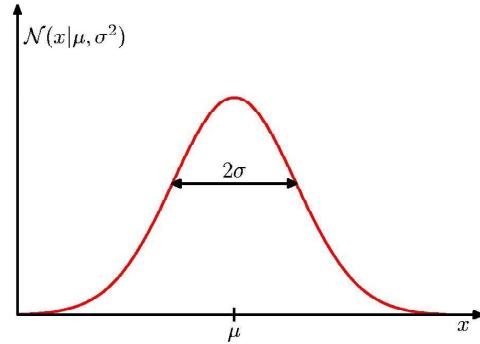
Covariance  
(vector  $\mathbf{x}, \mathbf{y}$ )

$$\begin{aligned} \text{cov}[\mathbf{x}, \mathbf{y}] &= \mathbb{E}_{\mathbf{x},\mathbf{y}} [\{\mathbf{x} - \mathbb{E}[\mathbf{x}]\}\{\mathbf{y}^T - \mathbb{E}[\mathbf{y}^T]\}] \\ &= \mathbb{E}_{\mathbf{x},\mathbf{y}}[\mathbf{x}\mathbf{y}^T] - \mathbb{E}[\mathbf{x}]\mathbb{E}[\mathbf{y}^T] \end{aligned}$$

# Univariate Gaussian Distribution

$$\mathcal{N}(x|\mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\}$$

Mean  
Variance

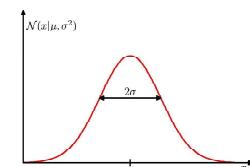


$$\mathcal{N}(x|\mu, \sigma^2) > 0$$

$$\int_{-\infty}^{\infty} \mathcal{N}(x|\mu, \sigma^2) dx = 1$$

# Gaussian Mean and Variance

$$\mathbb{E}[x] = \int_{-\infty}^{\infty} \mathcal{N}(x|\mu, \sigma^2) x dx = \mu$$



$$\mathbb{E}[x^2] = \int_{-\infty}^{\infty} \mathcal{N}(x|\mu, \sigma^2) x^2 dx = \mu^2 + \sigma^2$$

$$\text{var}[x] = \mathbb{E}[x^2] - \mathbb{E}[x]^2 = \sigma^2$$

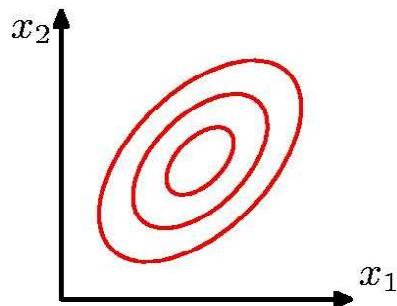
# Multivariate Gaussian Distribution

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\boldsymbol{\Sigma}|^{1/2}} \exp \left\{ -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right\}$$

Mean vector

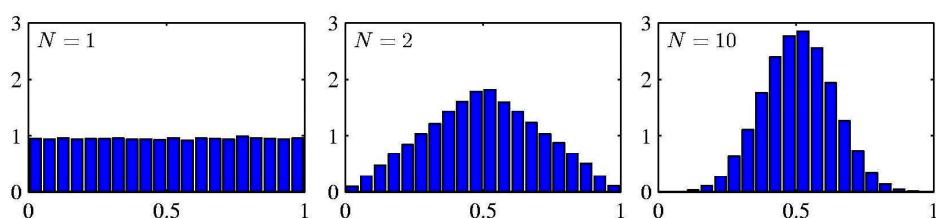
Covariance matrix

Space dimensionality



# Central Limit Theorem (Laplace)

- Sum of N random variables
  - Becomes increasingly Gaussian as N increases
    - (under some mild conditions)
- Example – N uniform [0,1] random variables:



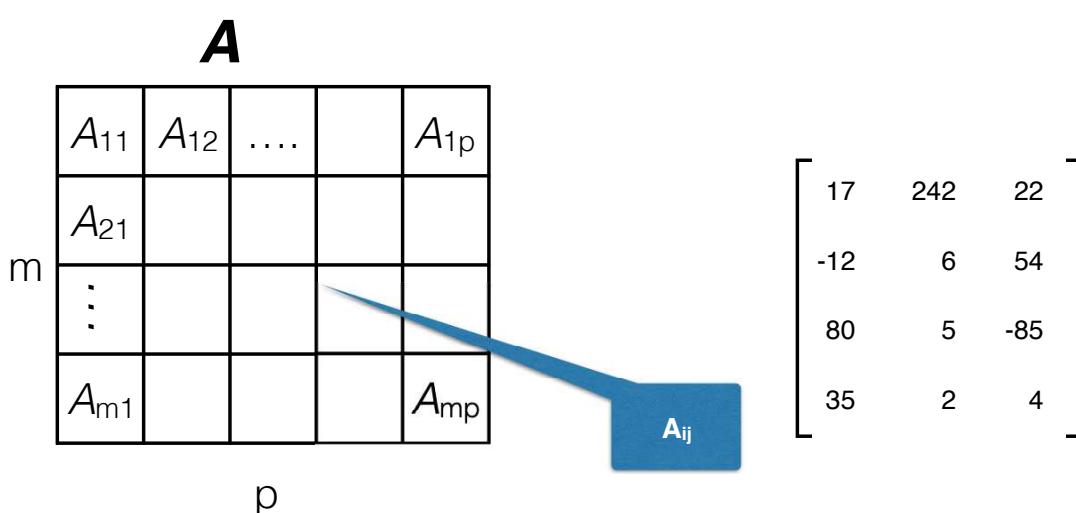
# This Lecture

Review selected basics of

- Probability theory
- Linear algebra
  - Matrices and vectors
  - Matrix-vector multiplication
  - Matrix-matrix multiplication
  - Matrix multiplication properties
  - Matrix inverse and transpose
  - Eigenvalues and eigenvectors



# Matrix



2D matrix:  
m rows, p columns

## Vector: Nx1 Matrix

$$V = \begin{bmatrix} 17 \\ -12 \\ 80 \\ 35 \end{bmatrix} \quad v = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \dots \\ \dots \\ \dots \\ v_n \end{bmatrix} \quad v = \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ \dots \\ \dots \\ \dots \\ v_{n-1} \end{bmatrix}$$

1-indexed or 0-indexed

## Matrix Addition

$$\begin{bmatrix} 4 & 10 \\ -3 & 20 \\ 10 & 2 \\ 1 & 8 \end{bmatrix} + \begin{bmatrix} 3 & 12 \\ -5 & 1 \\ 0 & 6 \\ 5 & -3 \end{bmatrix} =$$

$$\begin{bmatrix} 17 & 242 & 22 \\ -12 & 6 & 54 \\ 80 & 5 & -85 \\ 35 & 2 & 4 \end{bmatrix} + \begin{bmatrix} 3 & -5 \\ 12 & 2 \end{bmatrix} =$$

## Multiplying/Dividing by Scalar

---

$$\begin{bmatrix} 7 & 2 \\ -2 & 8 \\ 10 & -4 \end{bmatrix} \times 2 =$$

$$3 \times \begin{bmatrix} 7 & 2 \\ -2 & 8 \\ 10 & -4 \end{bmatrix} =$$

$$\begin{bmatrix} 8 & 20 \\ -20 & 4 \\ 40 & 0 \end{bmatrix} /4 =$$

## Combining Operations

---

$$\begin{bmatrix} 8 & 16 \\ -4 & 4 \\ 12 & -16 \end{bmatrix} /4 + \begin{bmatrix} 2 & 0 \\ 0 & 4 \\ -1 & 0 \end{bmatrix} \times 2 + \begin{bmatrix} 1 & 1 \\ 2 & 0 \\ -2 & 1 \end{bmatrix} =$$

# Today

---

Review selected basics of

- Probability theory
- Linear algebra
  - Matrices and vectors
  - Matrix-vector multiplication
  - Matrix-matrix multiplication
  - Matrix multiplication properties
  - Matrix inverse and transpose
  - Eigenvalues and eigenvectors



## Matrix-Vector Multiplication

---

$$A \times \vec{x} = \vec{y}$$

$$y_i = \sum_{k=1}^n A_{ik}x_k$$

$$\begin{bmatrix} & \\ & \end{bmatrix}_{m \times n} \times \begin{bmatrix} & \\ & \end{bmatrix}_{n \times 1} = \begin{bmatrix} & \\ & \end{bmatrix}_{m \times 1}$$

## Matrix-Vector Multiplication Examples

$$\begin{bmatrix} 2 & 3 \\ -1 & 1 \\ 0 & 4 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ -1 \end{bmatrix} =$$

$$\begin{bmatrix} 2 & 1 & 3 & 5 \\ 1 & -1 & 2 & 3 \\ 4 & 0 & -1 & 2 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ -2 \\ 1 \\ -2 \end{bmatrix} =$$

## Matrix-Vector Product and Prediction

Hypothesis as function of variable x, and parameters w

$$h(x) = \omega_0 + \omega_1 x$$

Hypothesis (prediction for given x)

$$h(x) = -5 + 0.1x$$

Different values of x

$$\begin{bmatrix} 1 & 10 \\ 1 & 20 \\ 1 & 30 \\ 1 & 40 \end{bmatrix} * \begin{bmatrix} -5 \\ 0.1 \end{bmatrix} = \begin{bmatrix} -4 \\ -3 \\ -2 \\ -1 \end{bmatrix}$$

# This Lecture

---

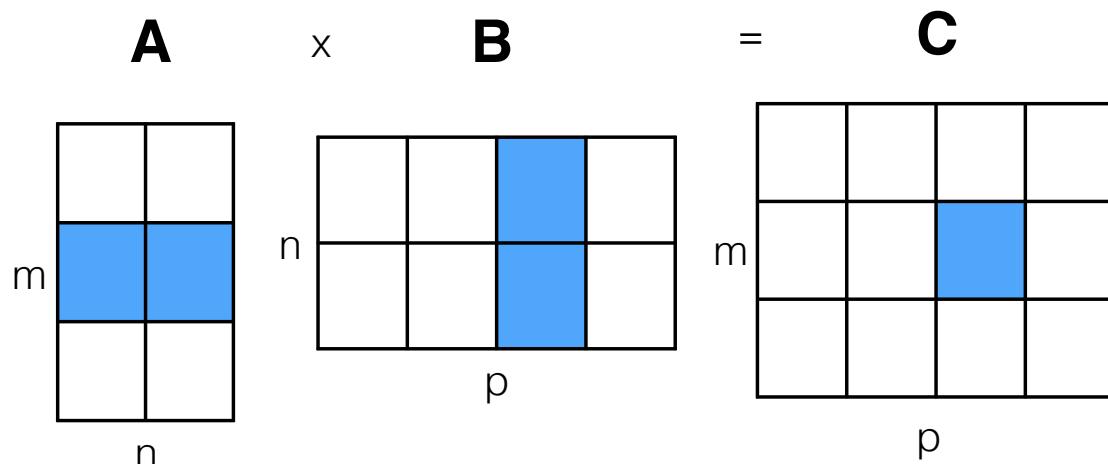
Review selected basics of

- Probability theory
- Linear algebra
  - Matrices and vectors
  - Matrix-vector multiplication
  - Matrix-matrix multiplication
  - Matrix multiplication properties
  - Matrix inverse and transpose
  - Eigenvalues and eigenvectors



## Matrix-Matrix Multiplication

---



Note matching dimension  $n$

# Matrix-Matrix Multiplication

---

$$A \times B = C$$

$$C_{ij} = \sum_{k=1}^n A_{ik} B_{kj}$$

$$\begin{bmatrix} & \\ & \\ m \times n & \end{bmatrix} \times \begin{bmatrix} & \\ & \\ & \\ n \times p & \end{bmatrix} = \begin{bmatrix} & \\ & \\ m \times p & \end{bmatrix}$$

## Matrix Multiplication Examples

---

$$\begin{bmatrix} 2 & 3 & 1 \\ -1 & 1 & 4 \\ 3 & -4 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 0 \\ -1 \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 3 & 1 \\ 3 & -4 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 1 \\ -1 \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 3 & 1 \\ 3 & -4 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 \\ -1 & 1 \\ 0 & -1 \end{bmatrix} =$$

# Linear Equations

$$\begin{aligned}2x - y &= 0 \\x + y &= 27\end{aligned}$$

$$\begin{bmatrix} 2 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 27 \end{bmatrix}$$

# Multiple Hypotheses

Hypothesis as function of variable  $x$ , and parameters  $w$

$$h(x) = w_0 + w_1 x$$

Three different hypotheses

$$\begin{aligned}h^1(x) &= -5 + 0.1x \\h^2(x) &= 80 + 2x \\h^3(x) &= 40 + 15x\end{aligned}$$

$x = \dots$

$$\begin{bmatrix} 1 & 10 \\ 1 & 20 \\ 1 & 30 \\ 1 & 40 \end{bmatrix}$$

$$\begin{bmatrix} -5 & 80 & 40 \\ 0.1 & 2 & 15 \end{bmatrix}$$

$$= \begin{bmatrix} -4 & 100 & 190 \\ -3 & 120 & 340 \\ -2 & 140 & 490 \\ -1 & 160 & 640 \end{bmatrix}$$

$h^1$     $h^2$     $h^3$

# This Lecture

---

Review selected basics of

- Probability theory
- Linear algebra
  - Matrices and vectors
  - Matrix-vector multiplication
  - Matrix-matrix multiplication
  - Matrix multiplication properties
  - Matrix inverse and transpose
  - Eigenvalues and eigenvectors



## Matrix Non-Commutativity

---

- In general, matrices do not commute

$$A \times B \neq B \times A$$

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 2 & 0 \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 1 \\ 2 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} =$$

- In some cases commutativity exists

# Matrix Associativity and Distributivity

- Matrix multiplication is associative

- $(AB)C = A(BC)$

- Matrix multiplication is distributive

- $A(B + C) = AB + AC$

- 

# Identity Matrix

- Denoted as  $I$  or  $I_{n \times m}$

$$I_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

- Example: 
$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

- For any matrix  $A$

- $A \cdot I = I \cdot A = A$

# This Lecture

---

Review selected basics of

- Probability theory
- Linear algebra
  - Matrices and vectors
  - Matrix-vector multiplication
  - Matrix-matrix multiplication
  - Matrix multiplication properties
  - Matrix inverse and transpose
  - Eigenvalues and eigenvectors



## Matrix Inverse

---

- If square matrix  $A$  has inverse then

$$A \cdot A^{-1} = A^{-1} \cdot A = I$$

- Singular or degenerate matrix — inverse does not exist
- Ex: Find inverse of 2x2 matrix
  -

# Matrix Invertibility

---

- A - square matrix
- A invertible iff
  - Determinant is non-zero
- A invertible iff
  - columns linearly independent
  - Equivalently — rows independent
- Cannot be inverted if has redundant rows/column (“linearly dependent”, “low rank”)

# Matrix Transpose

---

- If A is  $n \times m$  matrix and  $B = A^T$ , then

- B is  $m \times n$  matrix, and

$$B_{ij} = A_{ji}$$

- If  $y = Ax$ , then

$$y^T = (Ax)^T = x^T A^T$$

- $$\begin{bmatrix} 1 & 2 \\ -1 & 1 \\ 0 & -1 \end{bmatrix}^T =$$

# This Lecture

---

Review selected basics of

- Probability theory
- Linear algebra
  - Matrices and vectors
  - Matrix-vector multiplication
  - Matrix-matrix multiplication
  - Matrix multiplication properties
  - Matrix inverse and transpose
  - Eigenvalues and eigenvectors



## Eigenvalues and Eigenvectors

---

- Eigenvector  $x$  of matrix  $A$  —  $x$  lies on same line as  $Ax$ 
  - $Ax = \lambda x$
  - Scalar  $\lambda$  is eigenvalue
- Scalar  $\lambda$  is an eigenvalue of  $A$  iff  $(A - \lambda I)$  is singular
- Characteristic-polynomial equation for eigenvalues
  - $\det(A - \lambda I) = 0$
  - If  $A$  is  $n \times n$ , then above has  $n$  solutions (some may repeats)
- Finding eigenvectors
  - For each eigenvalue, solve  $(A - \lambda I)x=0$

# **Questions?**

---

# **Neural Networks and Deep Learning**

**Dr. Jerome J. Braun**

## **This Lecture: Introduction — Part 2**

Course: Neural Networks and Deep Learning  
IE 7615

---

---

**Unauthorized distributing, redistributing, posting,  
and/or reposting, of the materials of this course  
(including course lectures, lecture slides, slide-sets,  
syllabi, guidelines, assignments, quizzes, exams,  
presentations, software, electronic media, etc.),  
is prohibited.**

# This Lecture

---



- Deep learning – successes and potential
- Limitations ?
- Types of learning

## Application Areas (Recap)

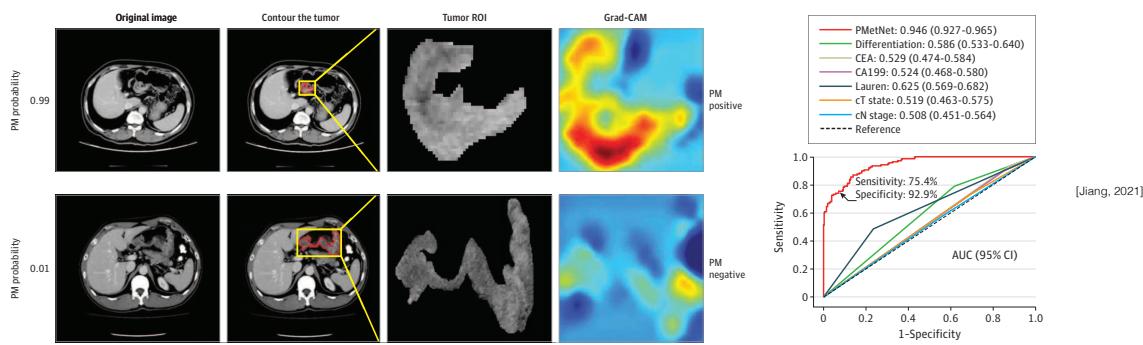
---

A few examples (out of many)	
Machine Vision	Speech & Language Processing (NLP)
Face Recognition, Object Recognition	Autonomous Vehicles
Self-driving cars	Robotics
Forecasting	Assistive Robotics
Weather, climate, etc.	Bioinformatics
Computational biology, neuroscience	“X-”informatics
Analytics (text, video, etc.)	Economics, Financial forecasting
Medical diagnostics	Insurance
Biomed data analysis; healthcare	Fraud detection
Drug development	Image/photo tagging
Personalized medicine	
... many others...	

- Span of potential applications enormous
- But beware of misapplication and overkills!

## Deep Neural Networks for Noninvasive Prediction of Occult Peritoneal Metastasis in Gastric Cancer (Recap)

- Jiang et al., JAMA Network Open, January 5, 2021
- Noninvasive preoperative (pre-surgery) assessment of occult peritoneal metastasis of gastric cancer
- Potentially useful to avoid unnecessary surgery and risk of associated complications
- CT imagery
- 1978 patients
- Densely connected convolutional neural network (CNN)
- Discrimination performance of network substantially higher than conventional clinicopathological factors

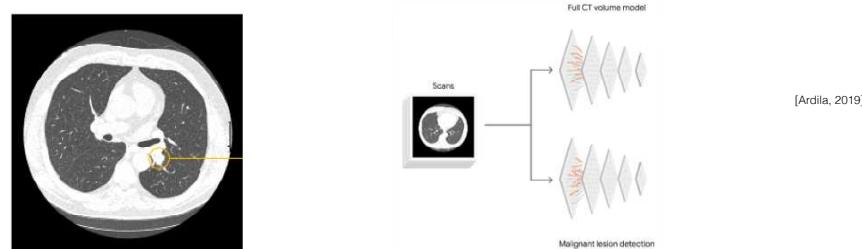


J. Braun

5

## Deep Neural Networks for Lung Cancer Screening (Recap)

- Ardila et al., Nature Medicine, May 2019
- Predict lung cancer risk by comparing patient's current and prior CT imaging
- Deep convolutional neural networks (CNN)
- 6,716 National Lung Cancer Screening Trial (NLST) cases
- 94.4% AUC performance
- Performance better or comparable with human readers (radiologists)

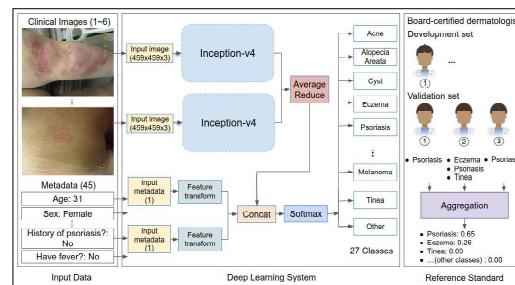


J. Braun

6

## Deep Neural Networks for Skin Lesion Classification (Recap)

- Liu *et al.*, 2019
- Differential diagnosis of 26 skin conditions from photographs and medical histories
- 14,021 development cases, 3,756 evaluation cases
- Variable number of deep convolutional neural network modules to process images
  - Inception-v4
- Shallow module for patient demographic information and medical history (metadata)



J. Braun

7

## Deep Neural Networks for Breast Cancer Screening (Recap)

- McKinney *et al.*, Nature, January 1, 2020
- Breast cancer prediction from mammograms
- Ensemble of three deep-learning models
- Exploit ImageNet, RetinaNet, ResNet
- All models implemented in TensorFlow
- Platform includes Google TPU hardware
- System performance — potential to perform better than trained radiologists
  - False positives reduction: 5.7% and 1.2% (US and UK)
  - False negative reduction: 9.4% and 2.7%
  - Able to generalize from UK data to US data
  - Able to outperform human experts
    - ♦ Six readers (radiologists)
    - ♦ AUC-ROC performance higher than that of human readers by 11.5% margin

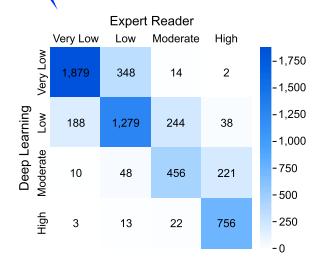
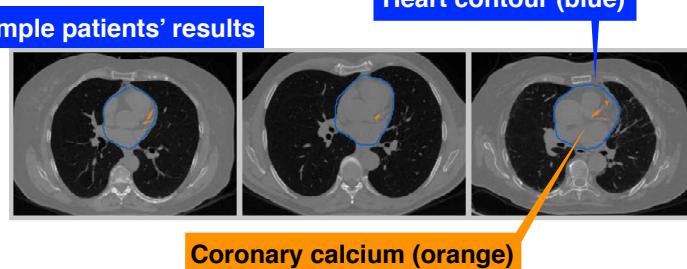
J. Braun

8

# Deep Neural Networks for Cardiovascular Risk Prediction (Recap)

- Zeleznik *et al.*, Nature Communications, Jan. 2021
- Coronary artery calcium — predictor of cardiovascular events
- Visible on all CT chest scans computed tomography (CT) scans
  - But quantification requires expertise, time, and specialized equipment
- Robust automatic quantification by deep-learning system
  - Convolutional neural networks
- 20,084 individuals from asymptomatic, and stable and acute chest pain cohorts
- High correlation of deep-learning system with quantification by expert readers
  - And robust test-retest reliability

## Example patients' results

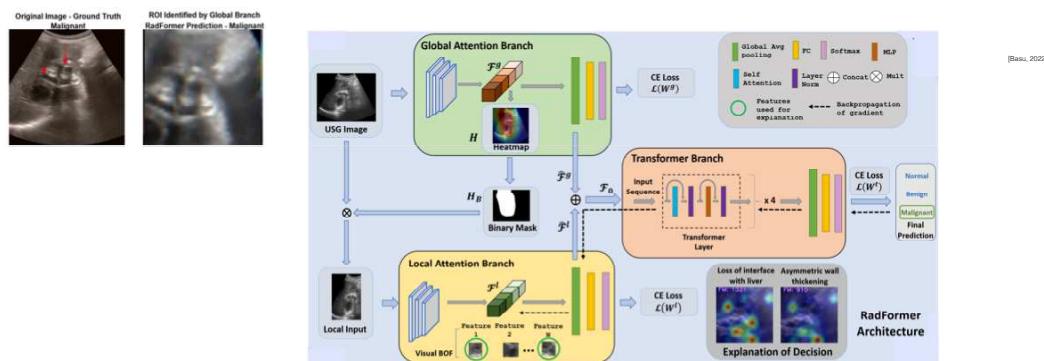


J. Braun

9

# Deep Neural Networks for Gallbladder Cancer Diagnostics (Recap)

- Basu *et al.*, 2022
- Diagnostics of gallbladder malignancies
- Input: ultrasound sonography images
- Transformer network architecture
- Basu *et al.* compared system results with conclusions of two expert radiologists
  - Found system performance was better

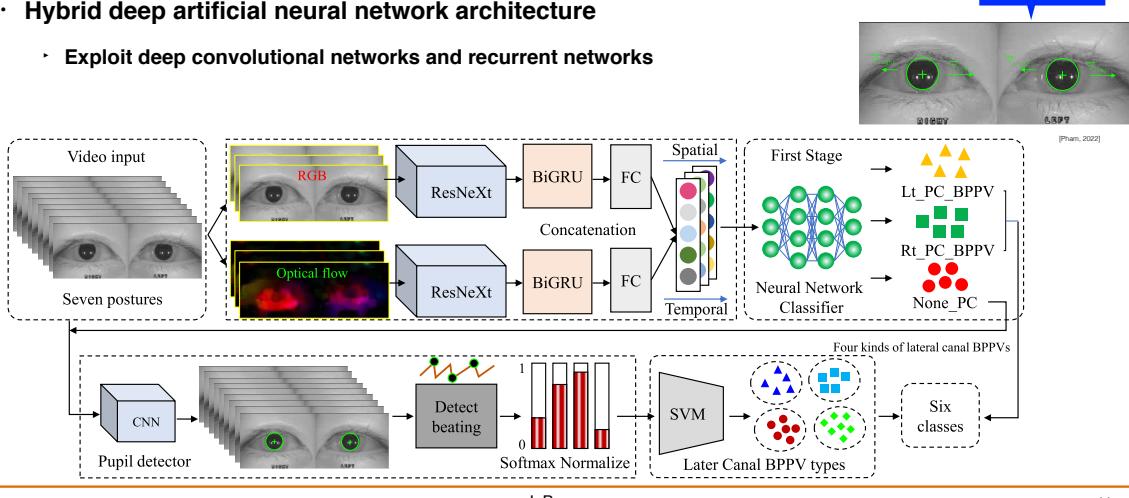


J. Braun

10

# Deep Neural Networks for BPPV Diagnosis (Recap)

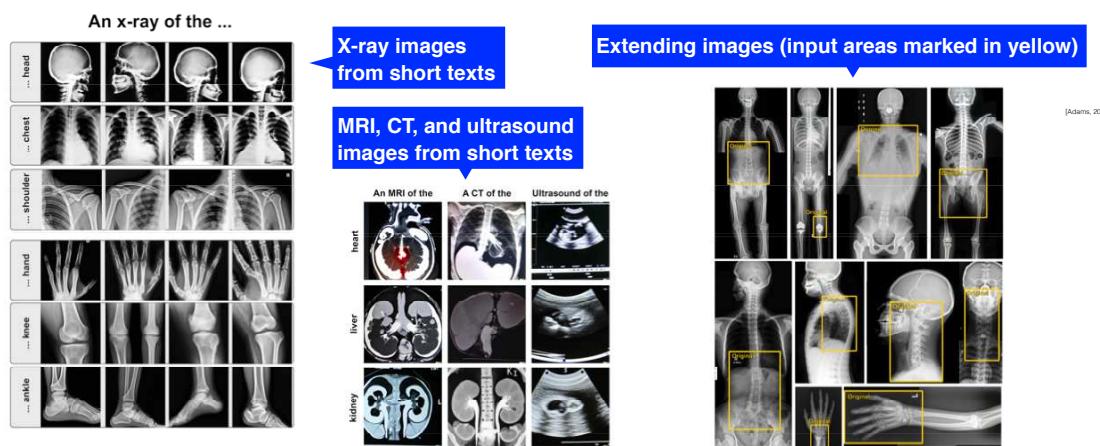
- Pham et al., Oct. 2022
- Diagnostics of benign paroxysmal positional vertigo (BPPV) types
  - Posterior canal types (Left, right), Lateral canal types: geotropic BPPV (left, right), apogeotropic (left, right)
- Input: video stream of patient eye-motion during diagnostic medical exam (Dix-Hallpike test)
- Hybrid deep artificial neural network architecture
  - Exploit deep convolutional networks and recurrent networks



11

# Deep Generative Models for Medical Imagery (Recap)

- Adams et al., 2022
- AI in radiology
  - Using DALL-E 2 generative model for text-to-image generation, image augmentation, and manipulation
  - DALL-E 2 learns relevant representations of X-ray images
    - + Zero-shot text-to-image generation of new images, continuation of image beyond original boundaries

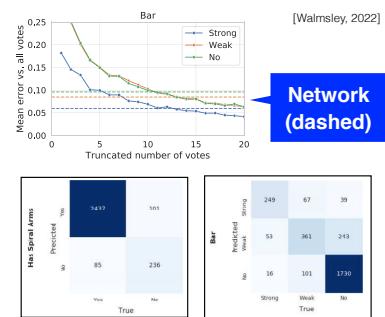


J. Braun

12

## Deep Neural Networks for Understanding Universe (Recap)

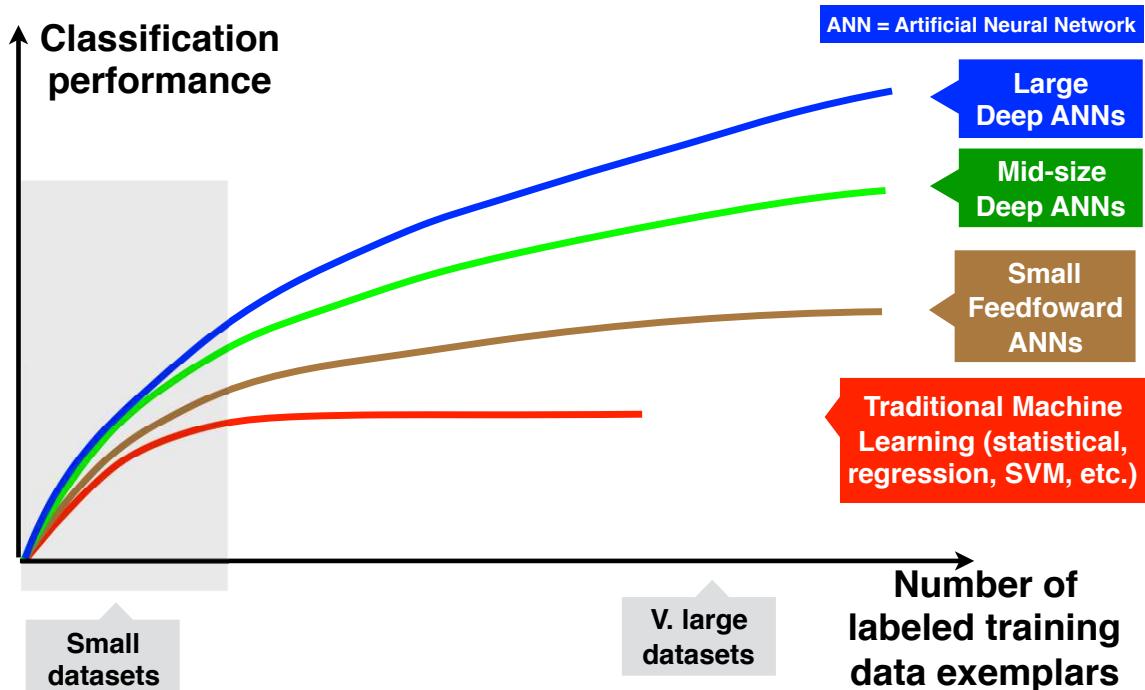
- Walmsley et al., MNRAS 509, 2022
  - Deep neural networks for astronomy and astrophysics research
  - Visual morphological classification of galaxies from images
    - + Morphology of galaxies— key to understanding galactic evolution
- Data — *Dark Energy Camera Legacy Survey* images of galaxies
- Ensemble of convolutional neural networks
  - Exploit EfficientNet-B0 architecture with modifications
- Predict morphology features of galaxies
  - E.g., spiral arms, bars, etc.
  - Measured against confident volunteer classifications
    - + Galaxy Zoo volunteers
  - Trained networks reach up to 99% accuracy
    - + Measured against ~10 volunteers, could be viewed as achieving superhuman performance



J. Braun

13

## Deep Learning vs. Other Machine Learning Paradigms



J. Braun

14

## Deep Learning / Artificial Intelligence

### What might be ahead

---

- Progressively more robust and versatile learning (reasoning)
- Ability to decide and act to accomplish goals
  - Leading to further questions and challenges, e.g., which goals, whose goals?
- Exhibit what we know as human emotions
  - At much higher level than today's *affective computing*
  - Different in their essence from today's *affective computing*
    - Although we do not yet know what this would entail since realm of emotions is not yet understood
- Ability to self-improve
  - Reorganize
  - Self-redesign

## This Lecture

---

---



- Deep learning – successes and potential
- Limitations ?
- Types of learning

## Deficiencies or Limitations? Watson's Error

In 2011,  
IBM Watson system  
won “Jeopardy!” TV game  
against human champions

... but ...

- Category: “U.S. Cities”
- Answer: *“Its largest airport was named for a World War II hero; its second largest, for a World War II battle”*
- Rutter and Jennings: “What is Chicago?”
- Watson: “What is Toronto?”

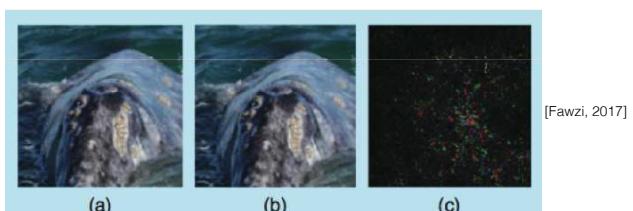
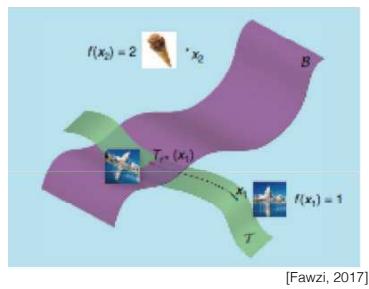
J. Braun

17

## Deficiencies or Limitations? Deep-Learning Robustness Issues

- Non-robustness of deep networks [Fawzi et al. 2017]

- Adversarial perturbations

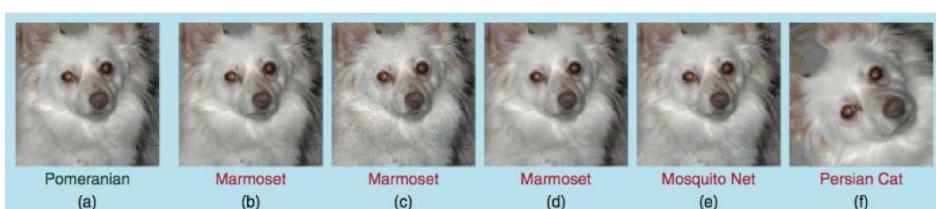


[Fawzi, 2017]

Original  
WHALE

Perturbed  
TURTLE

Adversarial  
Perturbation



[Fawzi, 2017]

J. Braun

18

# This Lecture

---

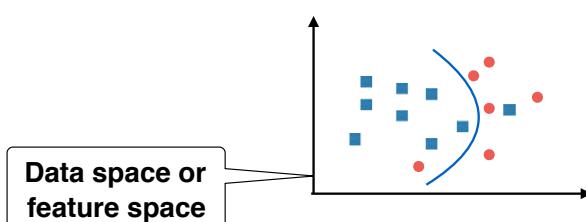
- › Deep learning – successes and potential
  - › Limitations ?
- 
- › Types of learning

# Machine Learning in a Nutshell

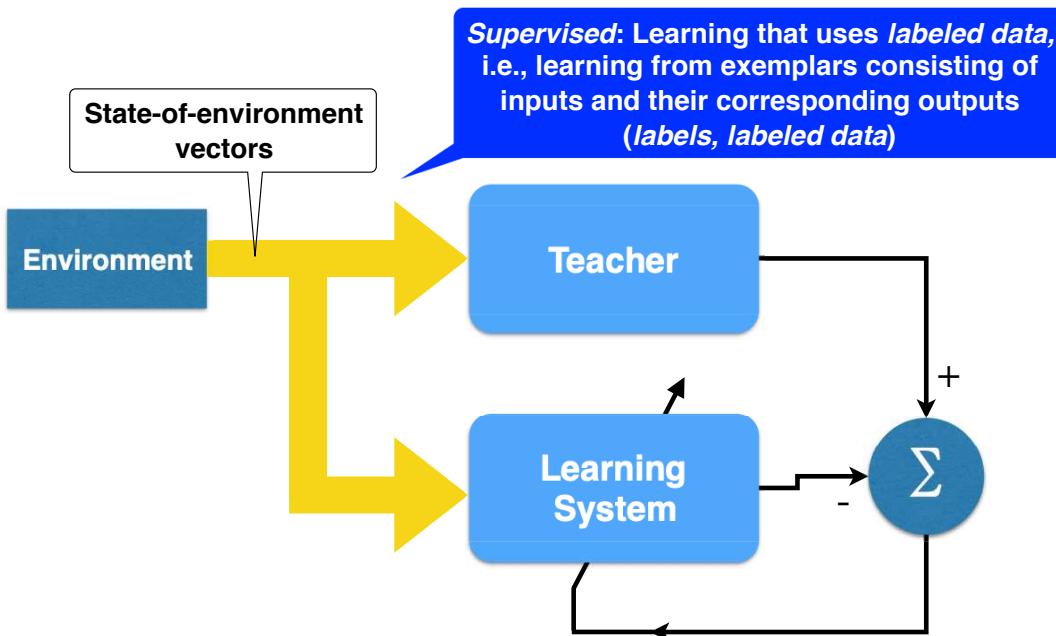
---

---

- “Typical” interpretation
  - › Derive decision boundary that “best” separates regions of different data-categories (classes of data)
- Statistical Learning Theory (Vapnik)
  - › Derive “recipe” for categorizing given data
- Features
  - › Functions of data
  - › Meant to represent data – salient aspects of data



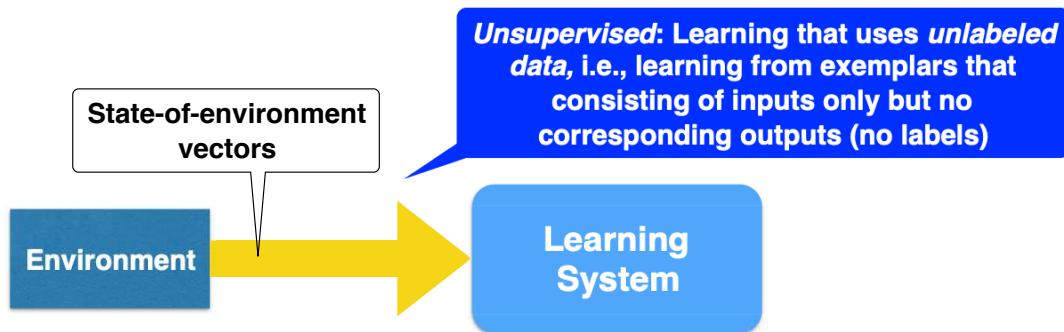
# Supervised Learning



# Supervised Learning

- Training data consist of inputs and outputs
  - Training set
- Previously unseen (novel) input to be categorized
  - Test set
- Categorize/classify = determine output given input
- Output types
  - Discrete (categorical) — classification task
  - Continuous — regression task

# Unsupervised Learning

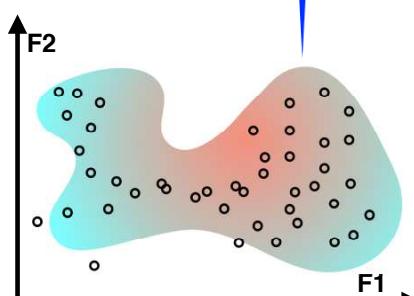
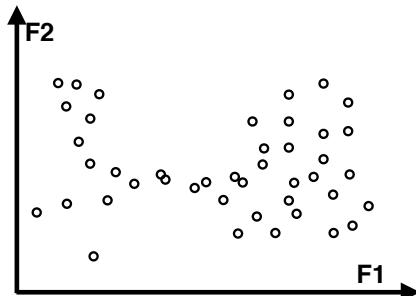


J. Braun

23

# Unsupervised Learning Tasks

- Training set contains data without labels
- Learning — from unlabeled data, learn some useful characteristics of data or process that underlies unlabeled data
  - Learn explicitly probability-distribution underlying dataset
  - Learn to generate new data (synthesis)
  - Learn to remove noise from data — *denoising*
  - Determine clusters (“sub-groups”) in data — *clustering*
  - etc.

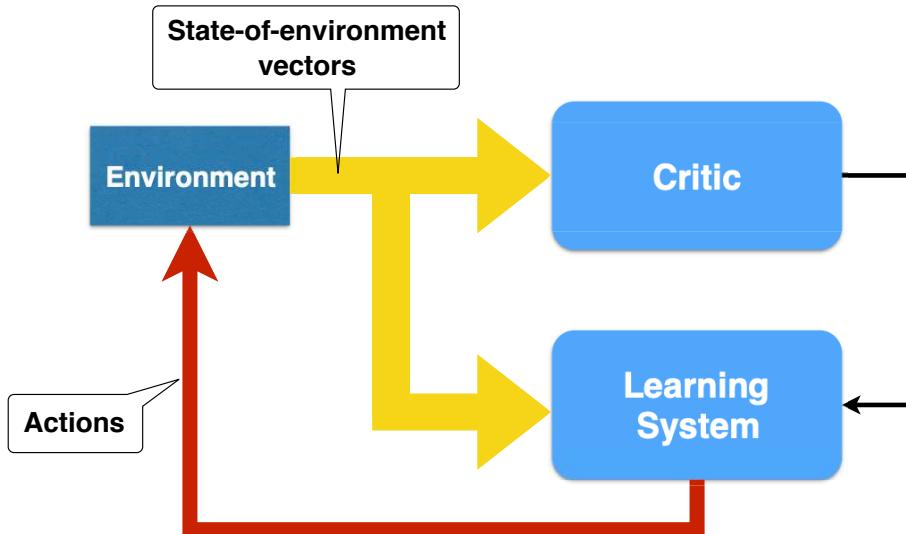


Underlying probability distribution

J. Braun

24

# Reinforcement Learning

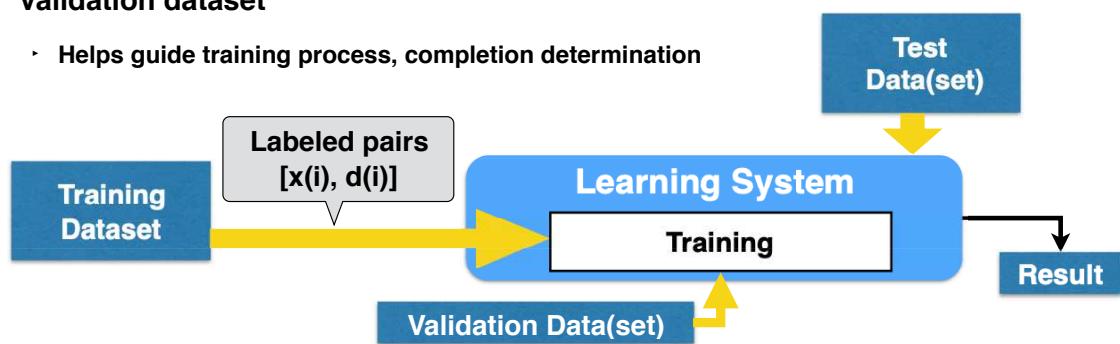


J. Braun

25

# Supervised Learning

- **Training — learning phase**
  - Training dataset — collection of labeled (annotated) exemplars
- **Testing — operation of trained system**
  - Test dataset — data to classify (categorize, recognize)
  - Labels used ONLY for performance assessment (not for classification)
- **Validation dataset**
  - Helps guide training process, completion determination

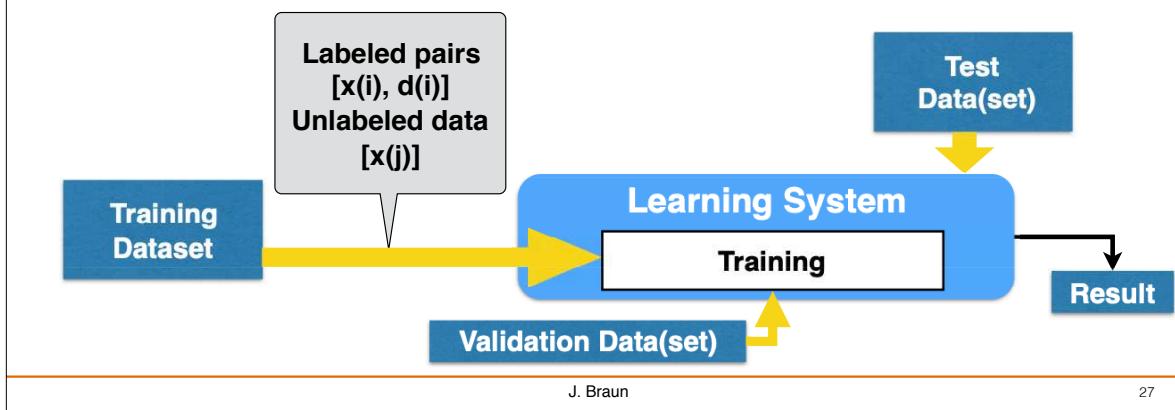


J. Braun

26

# Semisupervised Learning

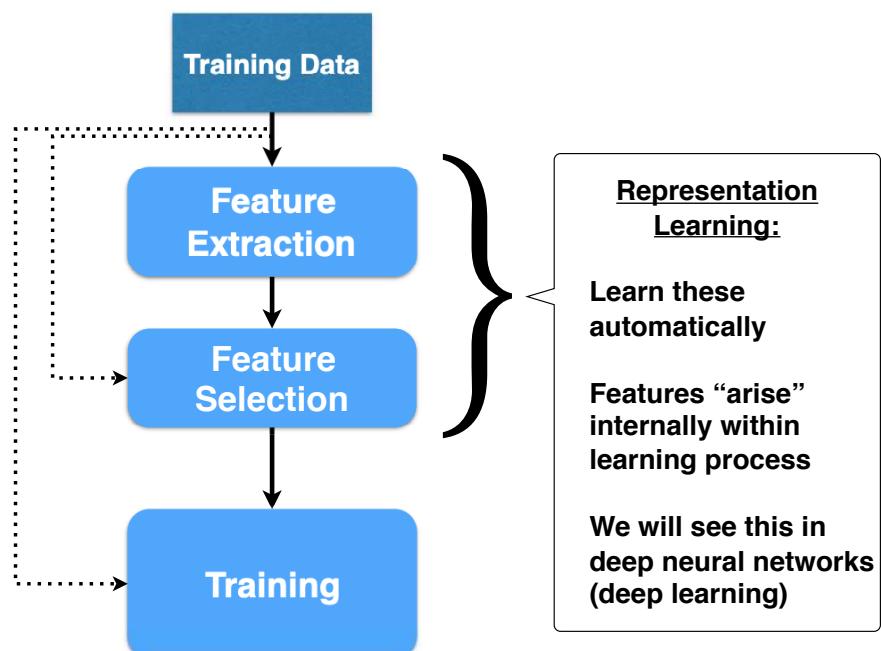
- Some training data annotated
  - Have category labels
- More training data available — but no labels
  - E.g., large collection of images, only some annotated



J. Braun

27

## Features



J. Braun

28

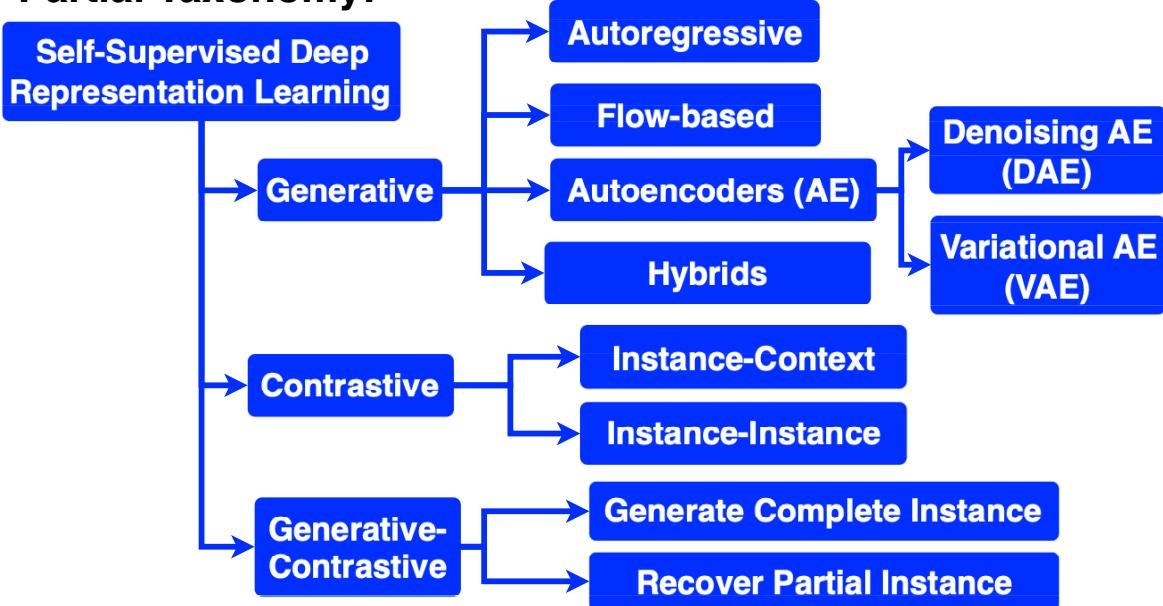
# Current Advanced Trends in Deep Learning

- Representation learning
- Learning from unlabeled data
  - Semi-supervised learning
  - Self-supervised learning — learning to classify from unlabeled data
- Transfer learning
  - Train on “pretext” (“proxy”) task
  - Transfer (apply) trained deep network to different tasks
    - Pre-train on one task
    - Fine-tune for another task
- Few-shot learning
  - Learn from few training instances
    - Two-shot learning, one-shot learning
  - Zero-shot learning

We will return to  
these later in  
this course

## Deep Self-Supervised Representation Learning

### Partial Taxonomy:



*More about this later in the course*

# Questions?

---

---

# **Neural Networks and Deep Learning**

**Dr. Jerome J. Braun**

## **This Lecture: Introduction — Part 3**

Course: Neural Networks and Deep Learning  
IE 7615

---

---

**Unauthorized distributing, redistributing, posting,  
and/or reposting, of the materials of this course  
(including course lectures, lecture slides, slide-sets,  
syllabi, guidelines, assignments, quizzes, exams,  
presentations, software, electronic media, etc.),  
is prohibited.**

# This Lecture

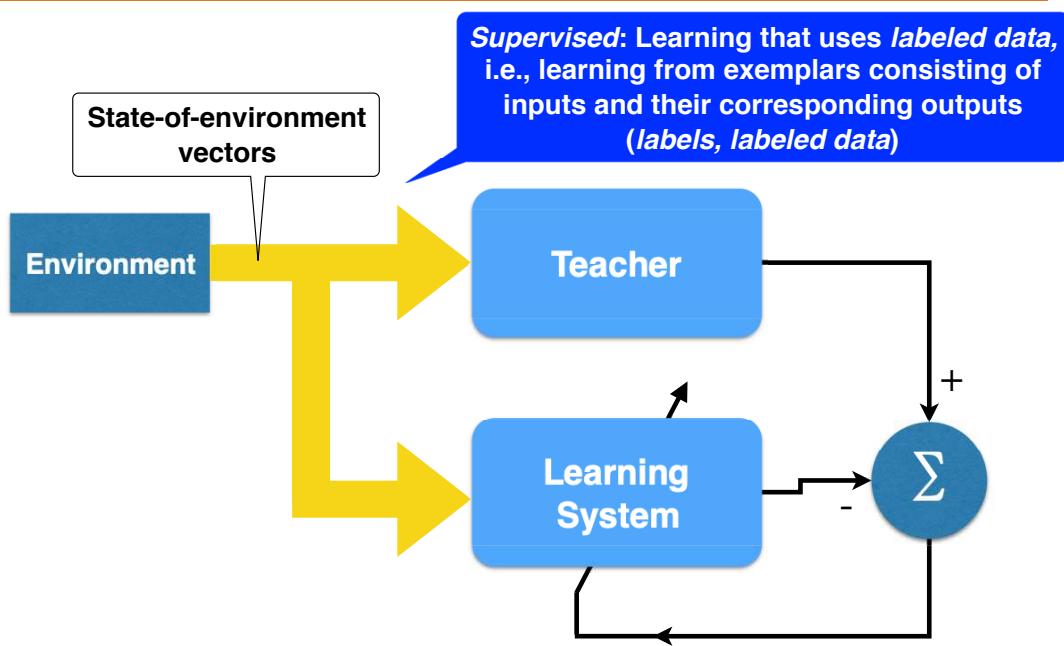
---



- Recap
- Learning – example
- Feature-space issues
- Decision boundary
- Multiple classes

## Supervised Learning (Recap)

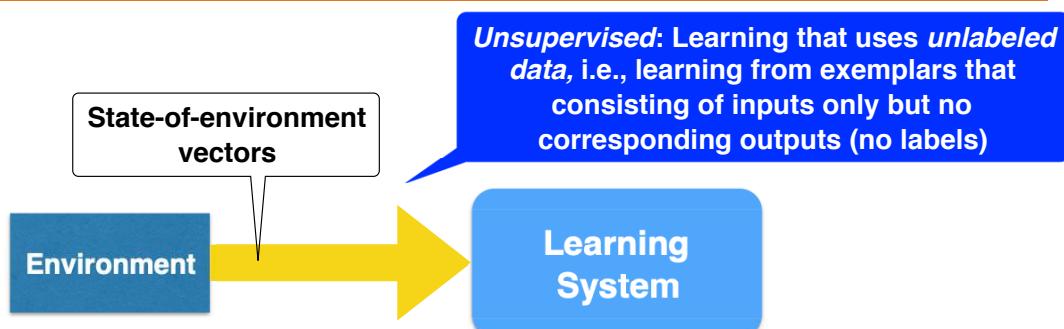
---



# Supervised Learning (Recap)

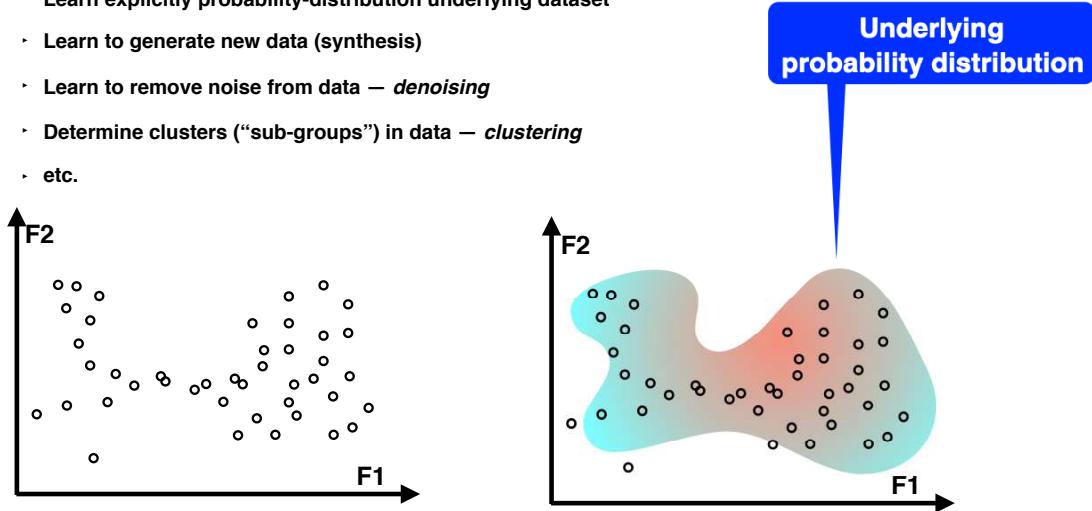
- Training data consist of inputs and outputs
  - Training set
- Previously unseen (novel) input to be categorized
  - Test set
  - Categorize/classify = determine output given input
- Output types
  - Discrete (categorical) — classification task
  - Continuous — regression task

# Unsupervised Learning (Recap)



# Unsupervised Learning Tasks (Recap)

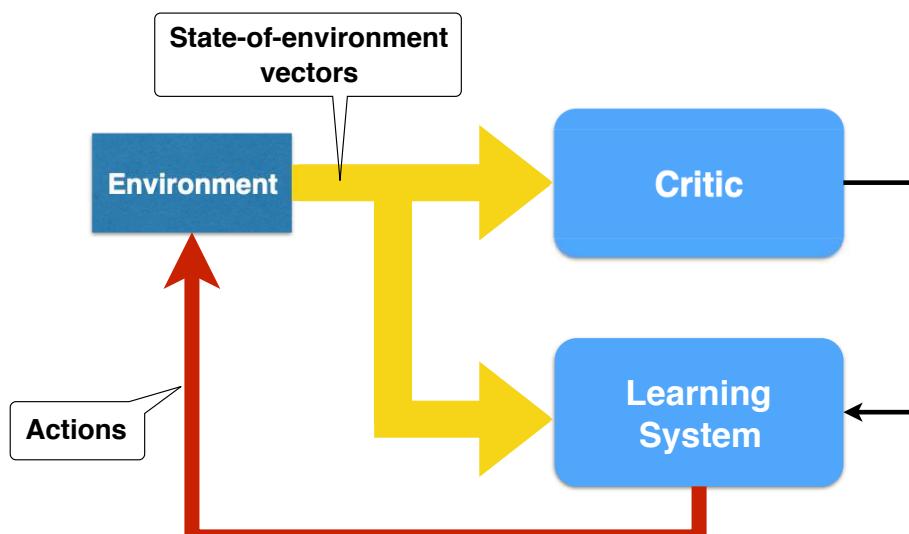
- Training set contains data without labels
- Learning — from unlabeled data, learn some useful characteristics of data or process that underlies unlabeled data
  - Learn explicitly probability-distribution underlying dataset
  - Learn to generate new data (synthesis)
  - Learn to remove noise from data — *denoising*
  - Determine clusters (“sub-groups”) in data — *clustering*
  - etc.



J. Braun

7

# Reinforcement Learning (Recap)



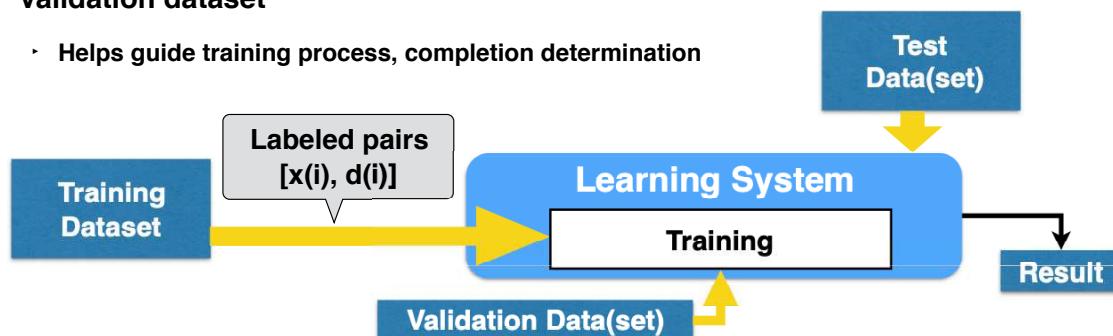
J. Braun

8

# Training, Validation, Testing (Recap)

In supervised learning

- Training — learning phase
  - Training dataset — collection of labeled (annotated) exemplars
- Testing — operation of trained system
  - Test dataset — unannotated data to categorize (classify, recognize)
- Validation dataset
  - Helps guide training process, completion determination

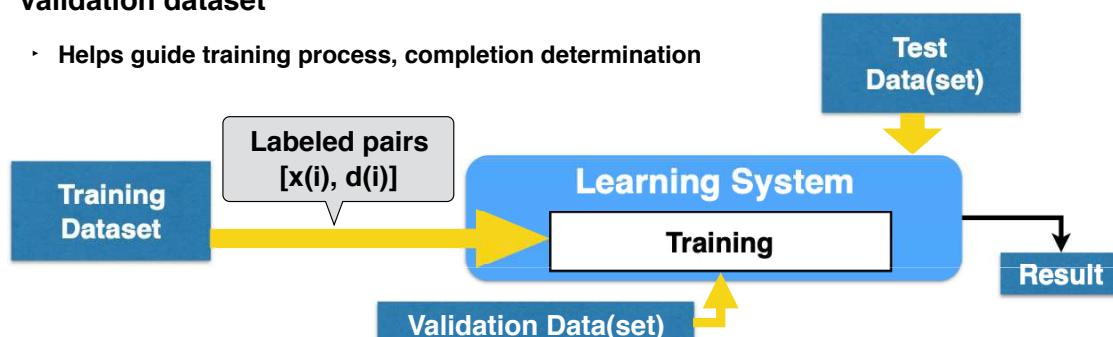


J. Braun

9

# Supervised Learning (Recap)

- Training — learning phase
  - Training dataset — collection of labeled (annotated) exemplars
- Testing — operation of trained system
  - Test dataset — data to classify (categorize, recognize)
  - Labels used ONLY for performance assessment (not for classification)
- Validation dataset
  - Helps guide training process, completion determination

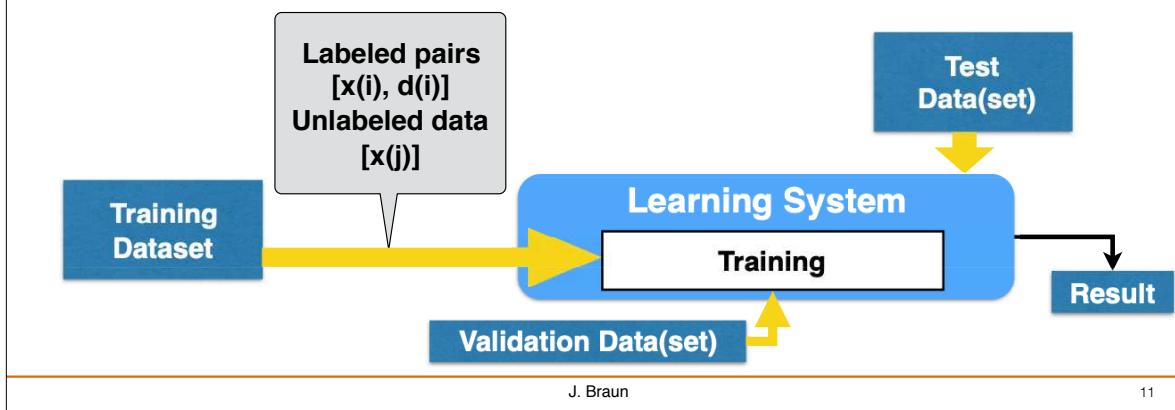


J. Braun

10

# Semisupervised Learning (Recap)

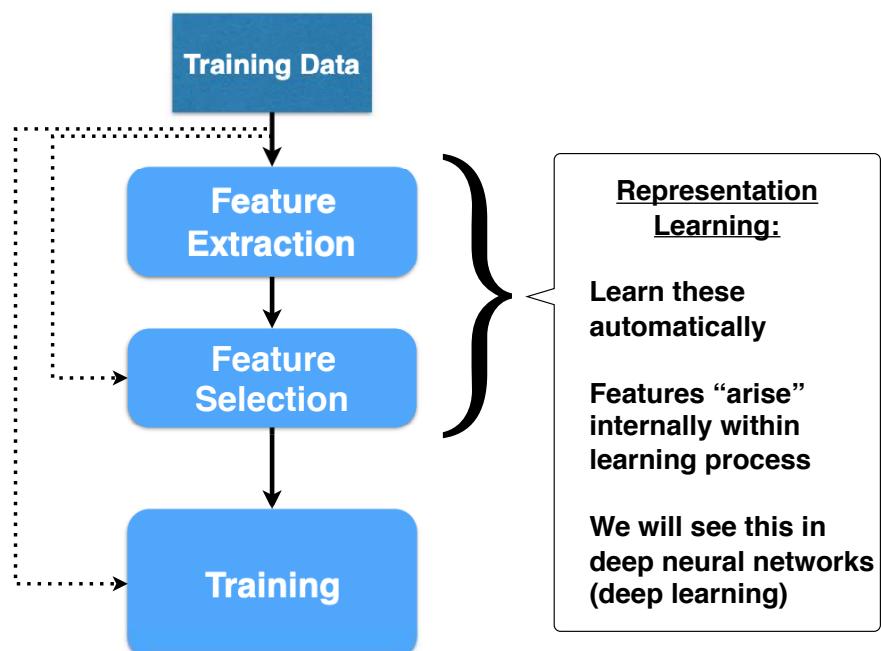
- Some training data annotated
  - Have category labels
- More training data available — but no labels
  - E.g., large collection of images, only some annotated



J. Braun

11

# Features (Recap)



J. Braun

12

# This Lecture

---



- Recap
- Learning – example
- Feature-space issues
- Decision boundary
- Multiple classes

J. Braun

13

## Hypothetical Example

---

- Automated preserves-making facility
  - Cherry preserves and grape preserves
- Video camera monitoring conveyor belt
  - Split fruits into two separate follow-up production lines

- 

Cherries



Grapes



[Images credit: Wikipedia, Wikipedia Commons]

J. Braun

14

# Distinguishable Characteristics

Cherries



Grapes



[Images credit: Wikipedia, Wikipedia Commons]

- Many distinguishable characteristics not available to camera, e.g., physical and chemical characteristics
- Potential image characteristics for “cherry/grape problem”
  - Brightness (or color)
  - Size
  - Shape (e.g., circular or not)
  - ...

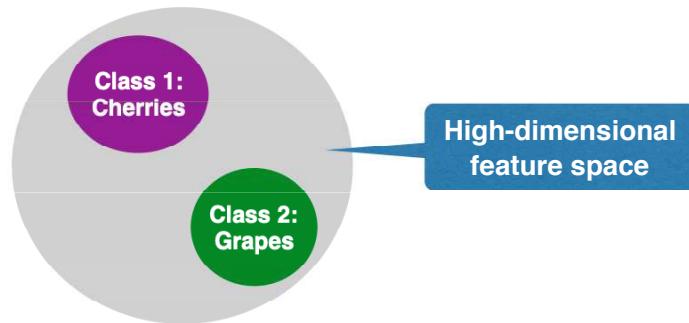
Deep neural networks often operate on raw inputs (e.g., images) directly

- Without human-engineered features
- Features arise (emerge) within network (*representation learning*)

(As we will see when we study deep networks)

# Ideal vs. Realistic Feature Space

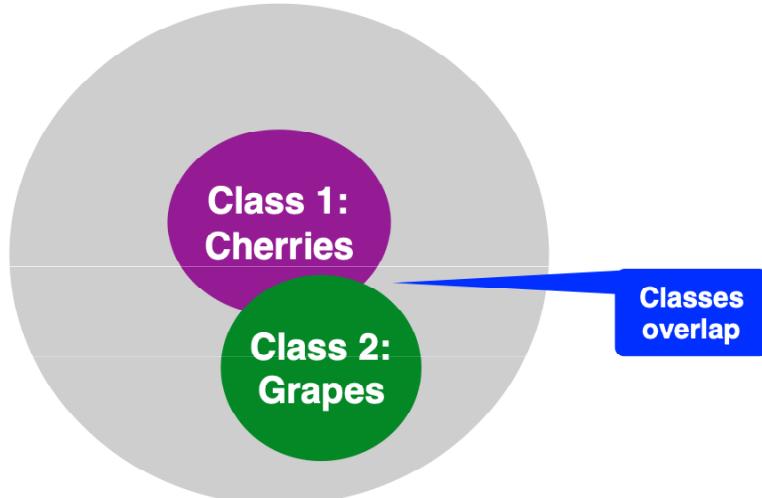
- If features represented entire characteristics, feature space would be separable
  - High-dimensional feature space



- But many features not accessible to available sensing modality such as video camera (image domain)

# Ideal vs. Realistic Feature Space

- In many subspace projections – e.g., image domain
  - Classes not separable

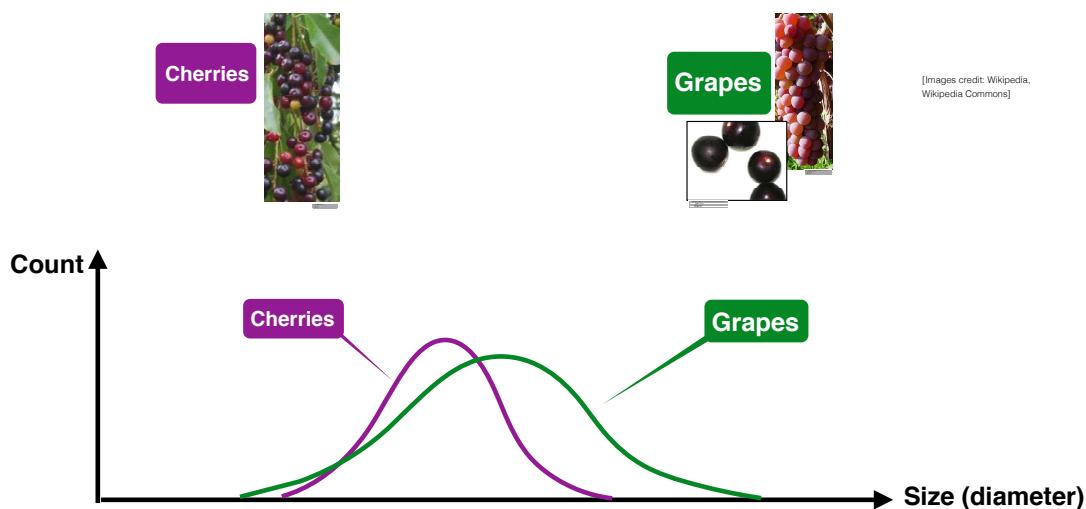


J. Braun

17

## Training Set — Size Feature

- Collect many examples (exemplars) of each type
- Generate marginal distribution of size feature

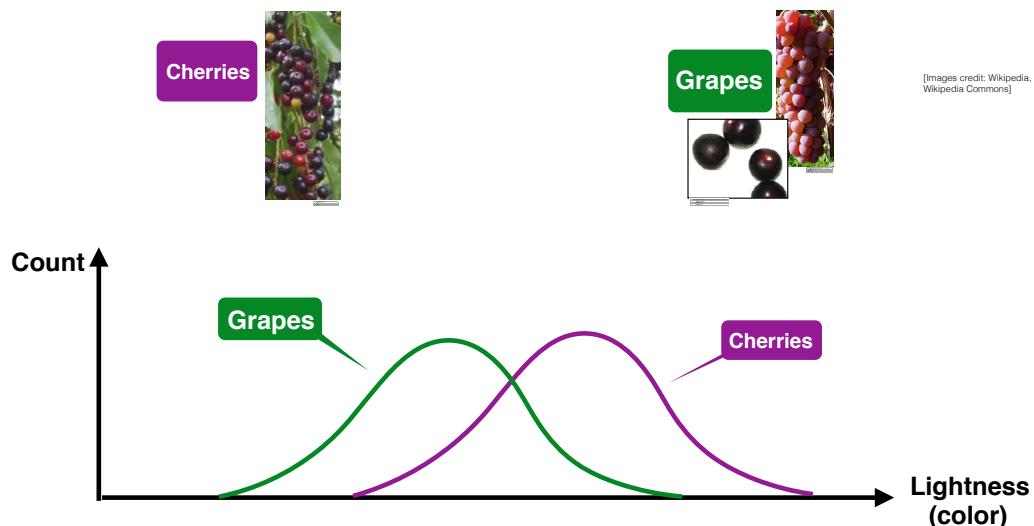


J. Braun

18

# Training Set – Lightness Feature

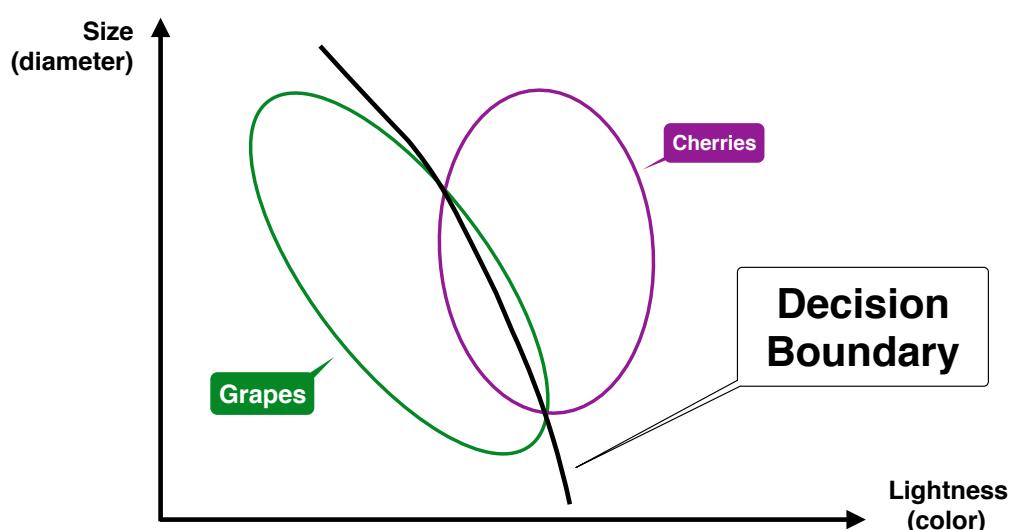
- Collect many examples (exemplars) of each type
- Generate marginal distribution of brightness feature



J. Braun

19

## Multidimensional Feature Space and Decision Boundary



- Joint distribution  $[\text{color}, \text{size}]^T$
- Two features may not be sufficient for sufficiently high accuracy
- Some points (fruits) misclassified

J. Braun

20

# This Lecture

---

- Recap
- Learning – example
- Feature-space issues
- Decision boundary
- Multiple classes



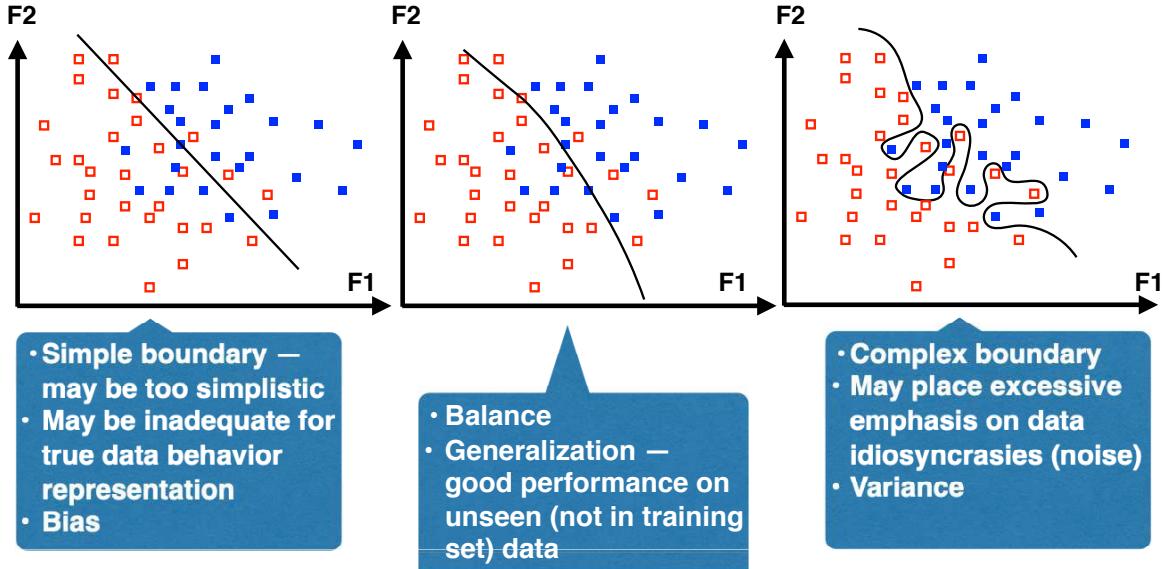
## Feature-Space Issues

---

- Issues associated with feature-design by humans (“feature engineering”)
  - What features, and how many (curse of dimensionality)
    - Feature extraction
  - Which features will work best
    - Feature selection
  - How mitigate feature redundancy and/or correlations
    - Dimensionality reduction
- Deep neural networks often do not require human-designed features
  - Representation learning in deep-learning constructs
    - Network operates with raw input data
    - Features are learned by network
      - Features or feature-equivalents arise (emerge) within network
  - More robust than human-designed features

We will see this when  
we study deep networks  
in detail in this course

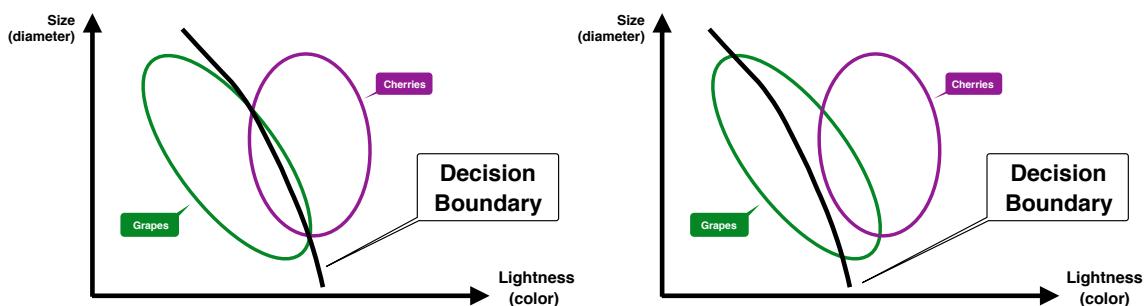
# Decision Boundary Complexity



J. Braun

23

# Misclassifications



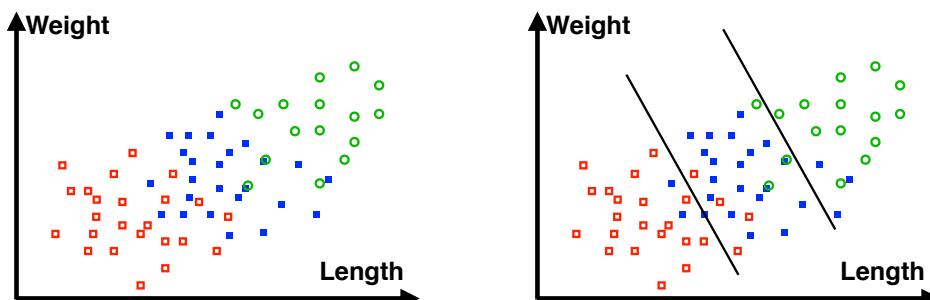
- Does it matter points of which class are misclassified ?
  - YES!
- Cost of errors may be different for one class than for another
  - E.g., misclassifying cherry as grape may be much more costly than vice-versa
    - Consider unexpected cherry-seeds in jars of grape jam (vs. a bit of grape-like taste of cherry jam)
- Decision boundary must be adjusted to account for different costs of errors — decision theory

J. Braun

24

## Three-Class Supervised Learning (Toy Example)

- Example: classify metal items, based on weight and size
  - Wrenches — generally longer and heavy
  - Bolts — generally longer or medium-length and light-weight or medium-weight
  - Nuts — short (small) and light-weight
- Training set — weight and length data of labeled items
  - Label for each item — e.g., W or B or N (“Wrench”, “Bolt”, “Nut”)
- Train classifier — learn to recognize Wrench vs. Bolt vs. Nut

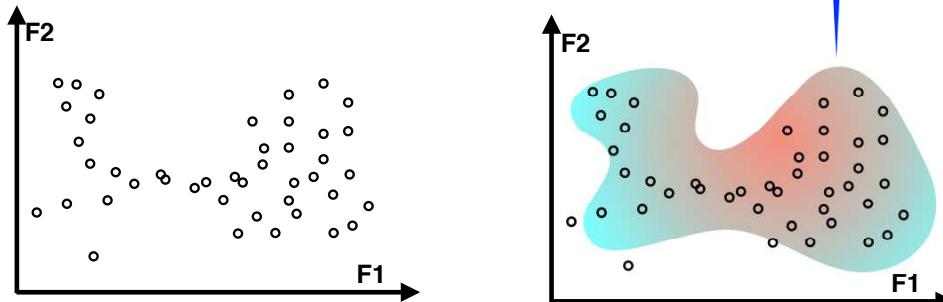


J. Braun

25

## Unsupervised Learning Tasks (Recap)

- Training set contains data without labels
- Learning — from unlabeled data, learn some useful characteristics of data or process that underlies unlabeled data
  - Learn explicitly probability-distribution underlying dataset
  - Learn to generate new data (synthesis)
  - Learn to remove noise from data — *denoising*
  - Determine clusters (“sub-groups”) in data — *clustering*
  - etc.



Underlying  
probability distribution

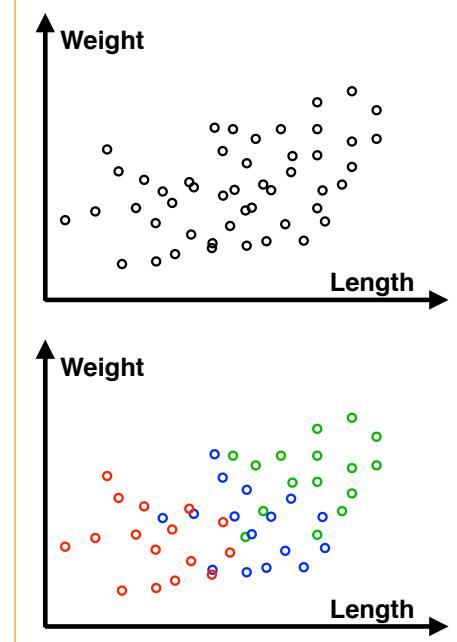
J. Braun

26

# Unsupervised Learning – Clustering Task

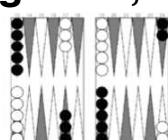
- Training set contains data without labels
- Learning
  - Discover internal “similarities” in training data
- At test-time
  - Decide based on “similarity” of test-exemplar to learned types (categories)
- Example: metal-parts dataset, consisting of weight and size data (features)

Modern unsupervised deep-learning techniques go beyond “classical” clustering



# Reinforcement Learning

- Given input or state
- Respond, predict output (outcome)
- Receive score on response
  - “reward/punishment”
- Applicable e.g., to games, robotics tasks, etc.



[Kohl, 2004]

## Determination of Learning Type – Exercise

---

- Learn to determine longevity (time-to-failure) of mechanical components, based on their physical features, such as thickness, material composition, etc.
- Learn to determine moves in card game
- Learn to determine driver competency/state, based on vehicle behavior in traffic
- What, and how, can be learned from large number of images showing several kinds of shirts

Questions?

---

# **Neural Networks and Deep Learning**

**Dr. Jerome J. Braun**

## **This Lecture: Supervised-Learning Basics I**

Course: Neural Networks and Deep Learning  
IE 7615

---

---

**Unauthorized distributing, redistributing, posting,  
and/or reposting, of the materials of this course  
(including course lectures, lecture slides, slide-sets,  
syllabi, guidelines, assignments, quizzes, exams,  
presentations, software, electronic media, etc.),  
is prohibited.**

# This Lecture

---

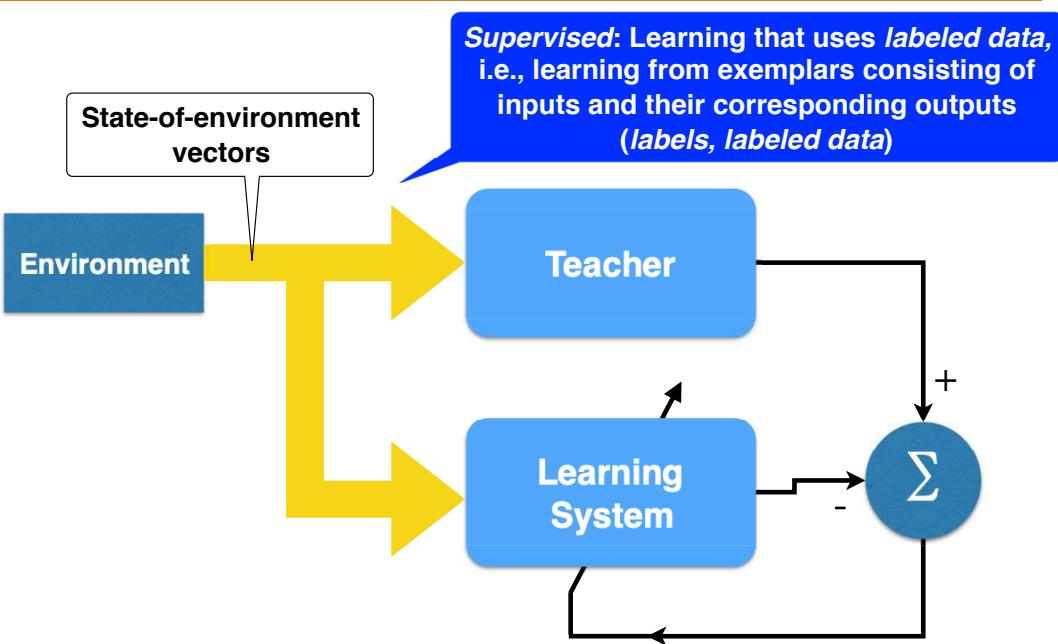
## → Linear Regression

Error (a.k.a. objective, cost, loss) function

Gradient descent (GD) basics

# Supervised Learning (Recap)

---



# Supervised Learning (Recap)

---

- Training data consist of inputs and outputs
  - Training set
- Previously unseen (novel) input to be categorized
  - Test set
  - Categorize/classify = determine output given input
- Output types
  - Continuous – regression task
  - Discrete (categorical) – classification task

## Linear Regression

---

- “Correct answers” given for each available data point
  - [data(i), desired\_value(i)]
- Regression task:
  - Given input
  - Predict (real-valued) output

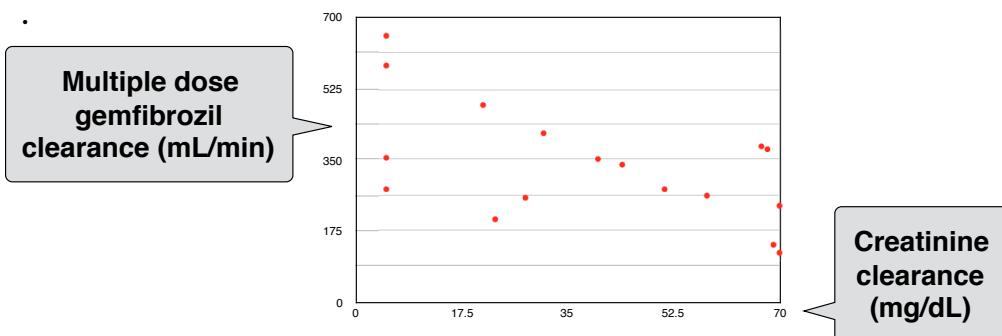
# Example Problem for Regression

- **Background**

- Serum creatinine clearance — measure of glomerular filtration rate (GFR) and useful for diagnosing renal disease

- **Issue**

- Prescribing gemfibrozil to patients with impaired renal function
  - Evans *et al.*, 1987 — study of impaired renal function and gemfibrozil clearance
  - Question: Does gemfibrozil remain longer in the body of patients with impaired renal function?



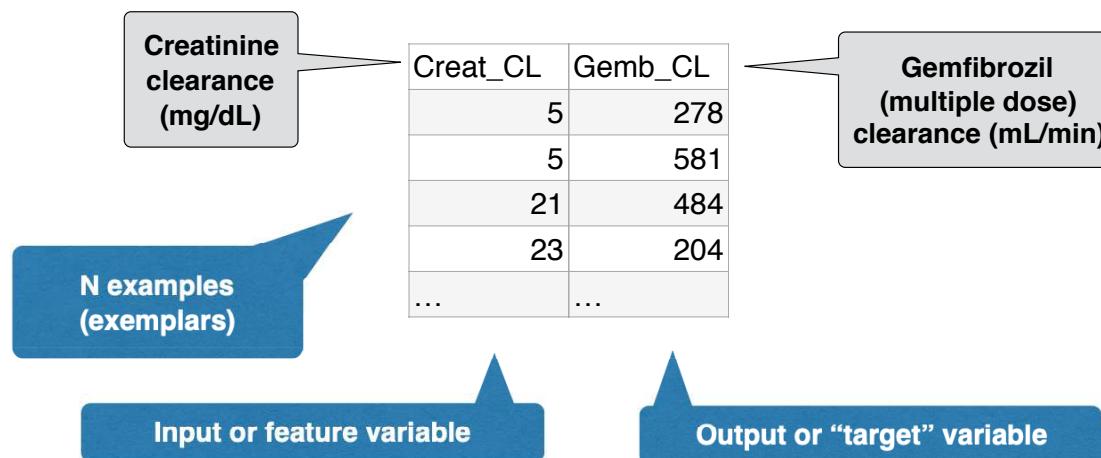
J. Braun

7

## Interpretation of Example

- Trivial to substitute many other specifics
- Instead of drug-clearance
  - Predict weight of item (e.g., hardware tool) from
    - ♦ Size
    - ♦ Size and appearance of material, etc.
  - etc.
- Task examples — suggestions?

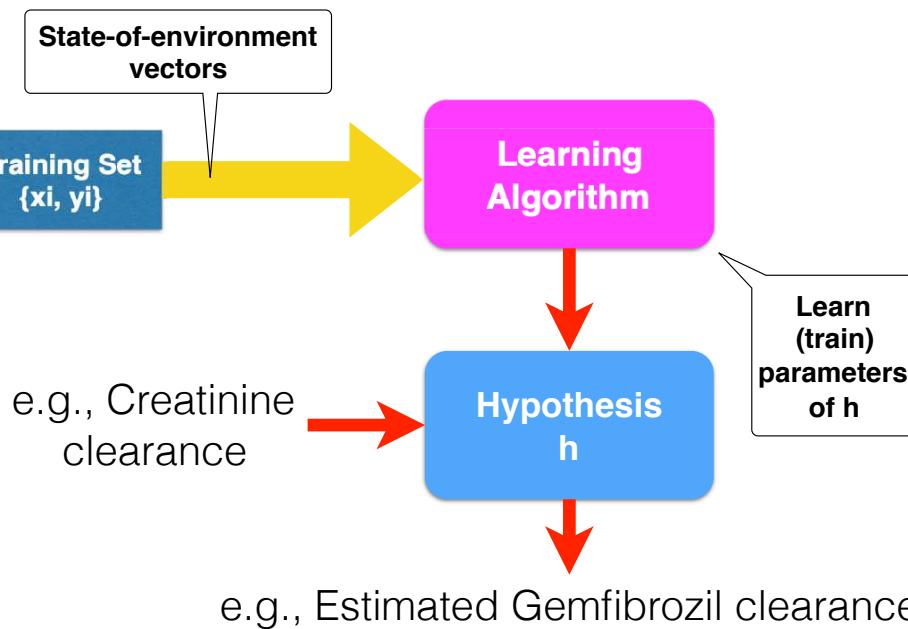
# Training Dataset



J. Braun

9

# Learning the Hypothesis Function

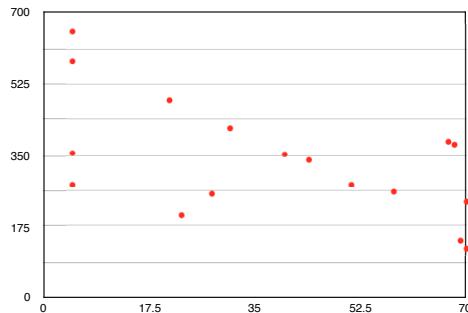


J. Braun

10

# Determining $h$

---



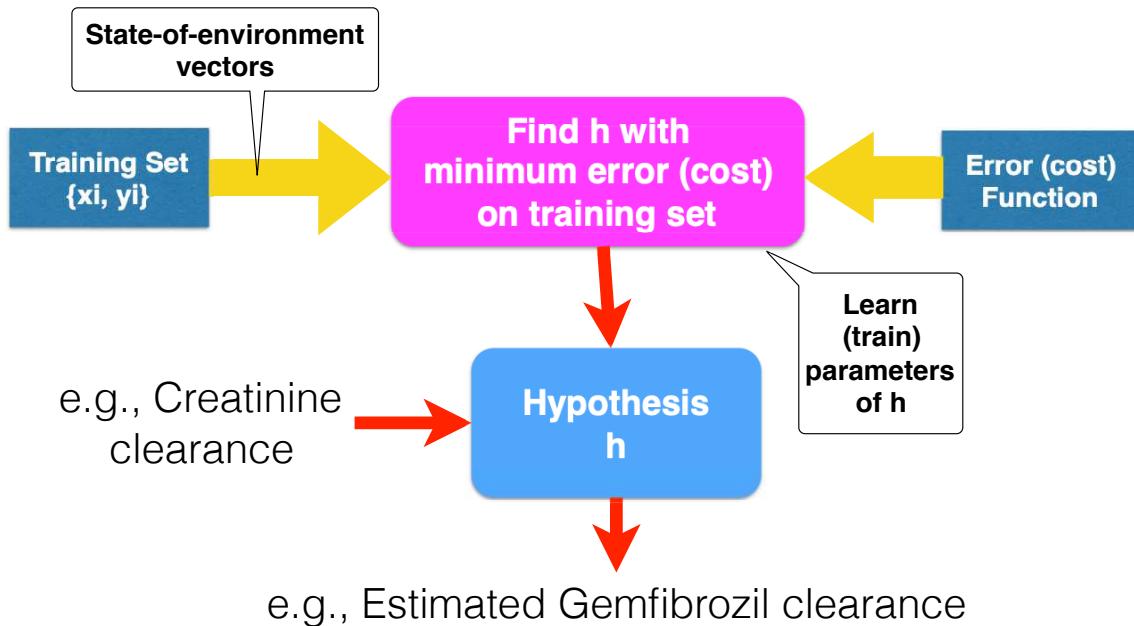
- **Linear hypothesis** 
$$h_w(x) = w_0 + w_1 x$$
  - **Parameters  $w_i$**
- **How do we determine “best” values of  $w$**

# This Lecture

---

- Linear Regression**
- **Error (a.k.a. objective, cost, loss) function**
- Gradient descent (GD) basics**

# Learning the Hypothesis Function



J. Braun

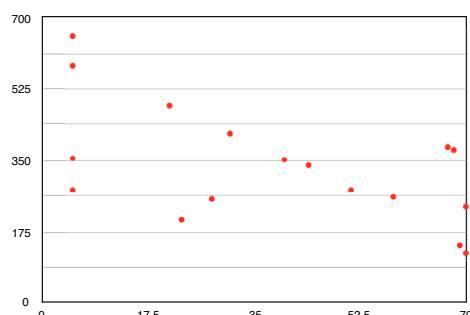
13

## Error Function For Linear Regression

- **Linear hypothesis**

$$h_w(x) = w_0 + w_1 x$$

- **Parameters  $w_i$**



- **What would be intuitively “good” error (cost) function**

- **Some measure of how actual values differ from desired values**

$$E(w_0, w_1) \leftarrow \text{Sum of } (\text{Actual} - \text{Desired})^2$$

i.e., sum of squared errors

... over all data points

**Objective:** minimize  $E(w_0, w_1)$

J. Braun

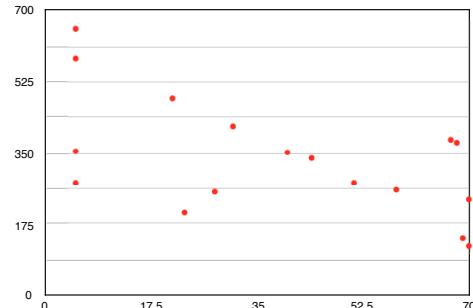
14

# Error Function For Linear Regression

- **Linear hypothesis**

$$h_w(x) = w_0 + w_1 x$$

- **Parameters  $w_i$**



- **What would be intuitively “good” error (cost) function**

- **Sum of squared errors (over all training dataset points)**

$$E(w_0, w_1) = \frac{1}{2N} \sum_{i=1}^N (h_w(x_i) - t_i)^2$$

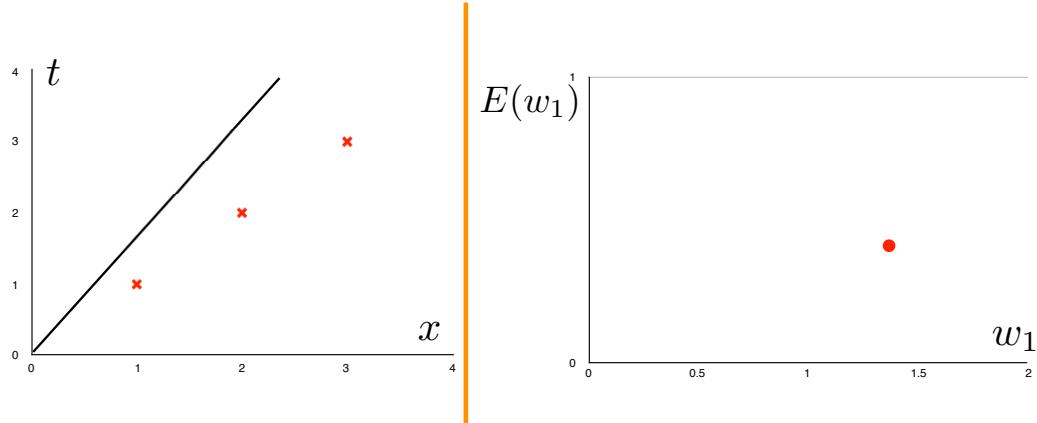
**Actual**      **Desired**

**Objective:**  $\underset{w_0, w_1}{\text{minimize}} E(w_0, w_1)$

## Choosing Parameters

$$h_w(x) = w_0 + w_1 x$$

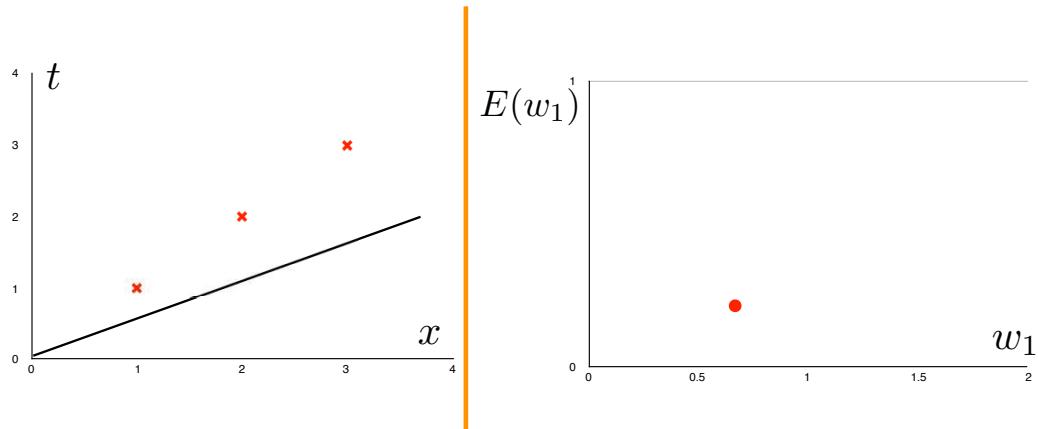
Focus for now on  $w_1$



$$E(w_0, w_1) = \frac{1}{2N} \sum_{i=1}^N (h_w(x_i) - t_i)^2$$

# Choosing Parameters

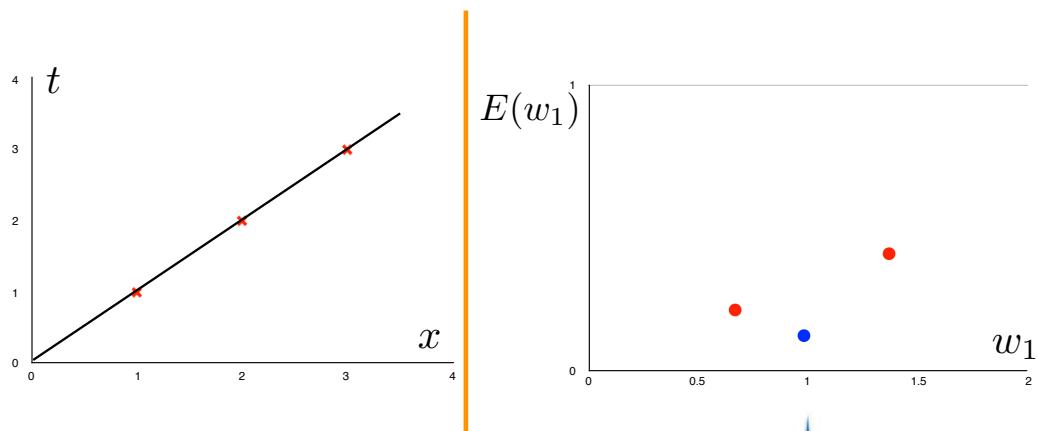
$$h_w(x) = w_0 + w_1 x$$



$$E(w_0, w_1) = \frac{1}{2N} \sum_{i=1}^N (h_w(x_i) - t_i)^2$$

# Choosing Parameters

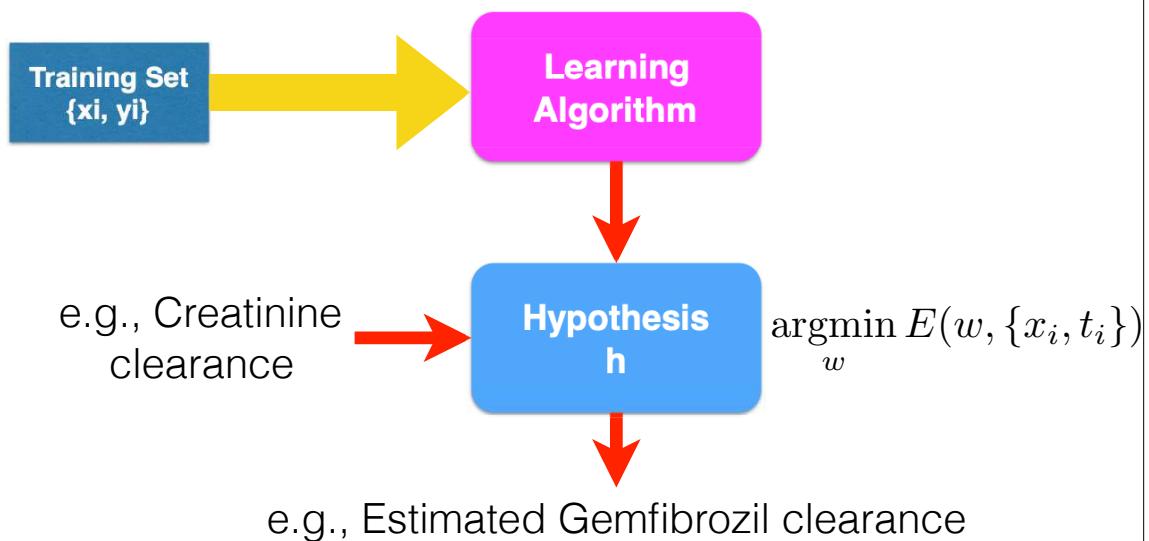
$$h_w(x) = w_0 + w_1 x$$



$$E(w_0, w_1) = \frac{1}{2N} \sum_{i=1}^N (h_w(x_i) - t_i)^2$$

Choose  $w_1$  for  
minimum error

## Supervised Learning of Hypothesis Function



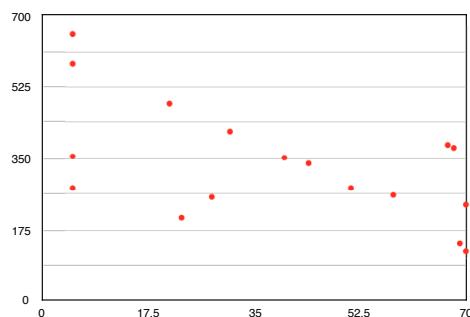
J. Braun

19

## Two-Dimensional Error Function

- **Two parameters ( $w_0, w_1$ )**

$$h_w(x) = w_0 + w_1 x$$



$$E(w_0, w_1) = \frac{1}{2N} \sum_{i=1}^N (h_w(x_i) - t_i)^2$$

**Objective:**  $\underset{w_0, w_1}{\text{minimize}} E(w_0, w_1)$

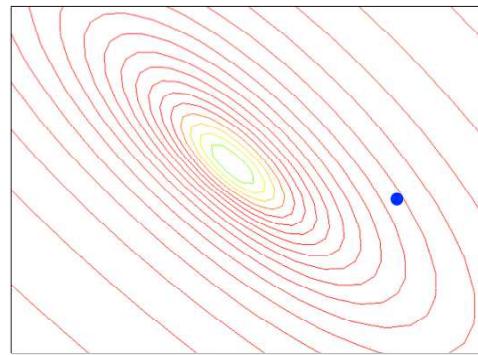
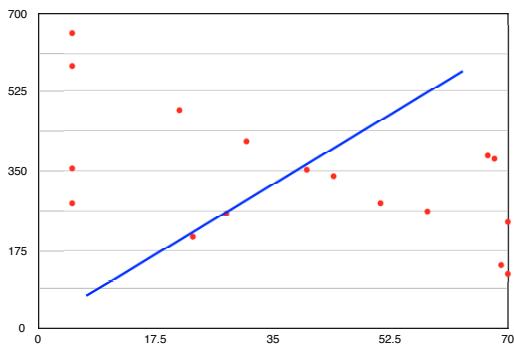
J. Braun

20

# Two-Dimensional Error Function

$$h_w(x) = w_0 + w_1 x$$

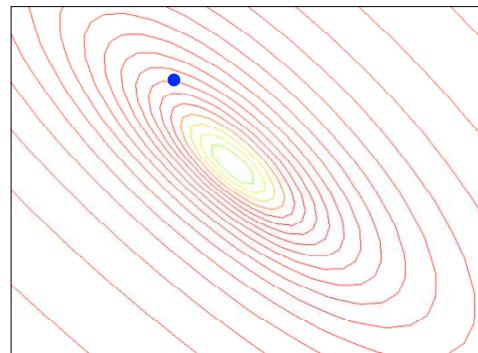
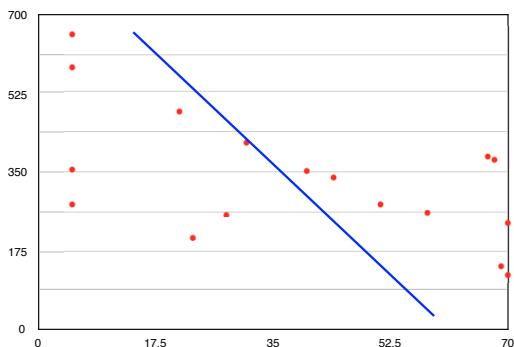
- Two parameters ( $w_0, w_1$ )



# Two-Dimensional Error Function

$$h_w(x) = w_0 + w_1 x$$

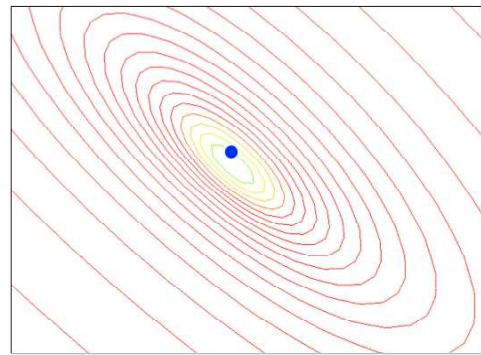
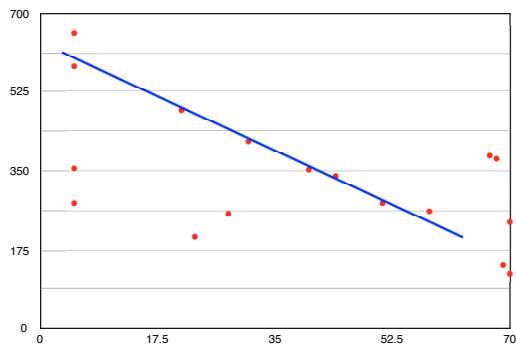
- Two parameters ( $w_0, w_1$ )



## Two-Dimensional Error Function

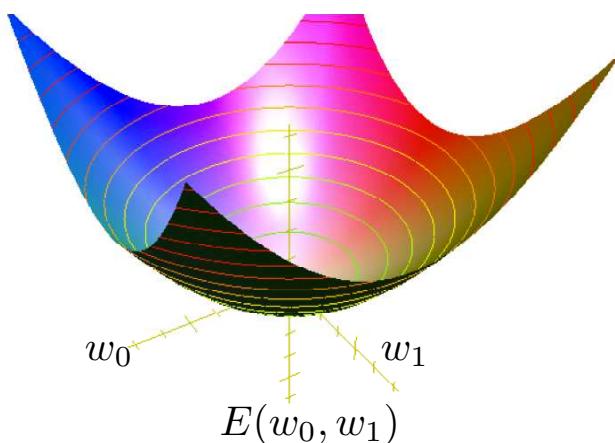
$$h_w(x) = w_0 + w_1 x$$

- Two parameters ( $w_0, w_1$ )



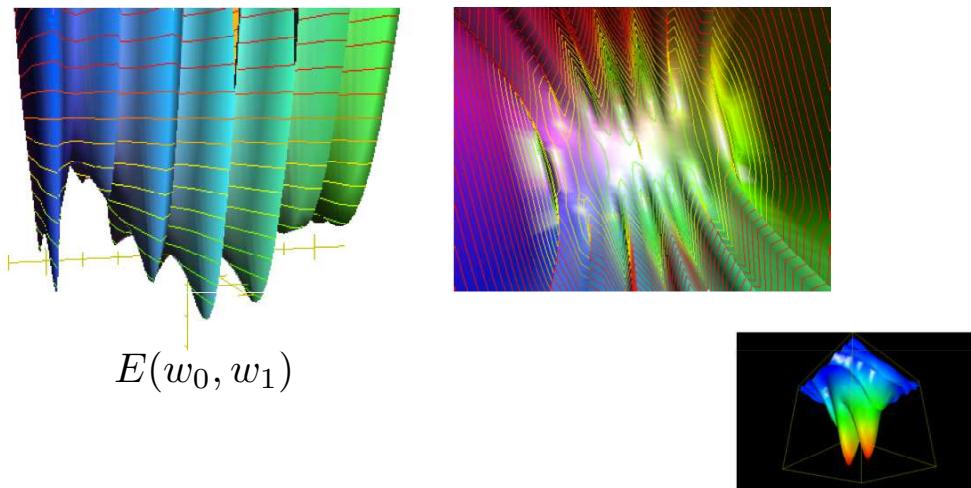
## Error Function Minimum

- For Linear Regression and least squares (SSD)
  - Quadratic dependence
- Unique minimum



# Non-monotonic Error Function Example

- In general, error functions may have multiple minima
  - Non-convex



J. Braun

25

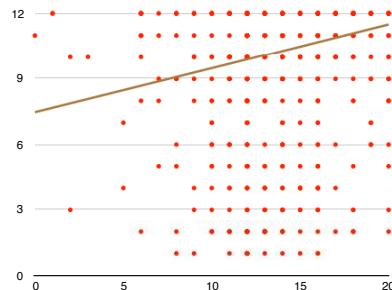
# Multiple Inputs — Multivariate Regression

- Example dataset — made from excerpt from GSS (General Social Survey) data

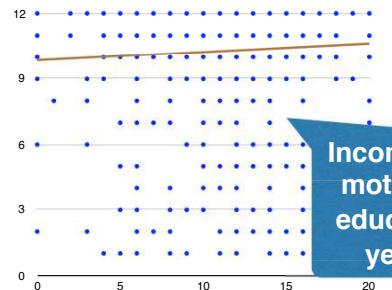
...	Age	Gender	Edud(y)	M_educ(y)	Income	...
...	...	...	...	...	...	...
...	41	1	20	19	12	...
...	51	2	20	19	12	...
...	22	2	12	18	2	...
...	20	2	13	18	2	...
...	53	1	14	18	3	...
...	29	2	16	18	4	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...

1 LT \$1000  
2 \$1000 TO 2999  
3 \$3000 TO 3999  
4 \$4000 TO 4999  
5 \$5000 TO 5999  
6 \$6000 TO 6999  
7 \$7000 TO 7999  
8 \$8000 TO 9999  
9 \$10000 - 14999  
10 \$15000 - 19999  
11 \$20000 - 24999  
12 \$25000 or more

Income vs.  
education  
years



Income vs.  
mother's  
education  
years



J. Braun

26

# Multiple Inputs – Multivariate Regression

Inputs/features (1 to n)						
...	k	k+1	k+2	k+3	...	...
...	...	...	...	...	...	...
...	41	1	20	19	...	12
...	51	2	20	19	...	12
...	22	2	12	18	...	2
...	20	2	13	18	...	2
...	53	1	14	18	...	3
...	29	2	16	18	...	4
...	...	...	...	...	...	...
...	...	...	...	...	...	...

Number of inputs

For now this is equivalent to saying “number of features”  
(it will not be the case later as we move forward in this course)

n

$x^{(i)}$

Inputs or features (vector) for i-th training dataset exemplar

$x_j^{(i)}$

Input or feature j for i-th training dataset exemplar

# Multivariate Linear Regression

- Multiple parameters w

$$h_w(x) = w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n$$

- Define  $x_0 = 1$

$$h_w(x) = w_0x_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n$$

- Allows writing h in matrix-equation form

- Error function

$$h_w(x) = \mathbf{w}^T \mathbf{x}$$

$$E(w_0, w_1, w_2, \dots, w_n) = \frac{1}{2N} \sum_{i=1}^N (h_w(x_i) - t_i)^2$$

**Objective:**  $\underset{w_0, w_1, \dots, w_n}{\text{minimize}} E(w_0, w_1, w_2, \dots, w_n)$

# Finding Error-Function Minimum

$$\underset{w_0, w_1, \dots, w_n}{\operatorname{argmin}} E(w_0, w_1, w_2, \dots, w_n)$$

$$\underset{\mathbf{w}}{\operatorname{argmin}} E(\mathbf{w})$$

- **Iterative solution**

- Start with initial value of  $\mathbf{w}$  — e.g., random guess
- Change  $\mathbf{w}$  to decrease error  $E(\mathbf{w})$
- Repeat until minimum reached
  - ♦ How do we know when we reach minimum?

Later in this course we will study  
much more sophisticated methods  
for initializing deep neural networks

- **Closed-form (direct) solution**

- Compute derivative of  $E(\mathbf{w})$
- Set to 0 w.r.t. parameters  $\mathbf{w}$
- For many error functions closed-form solution does not exist

Closed-form solutions  
are NOT available for  
deep neural networks

## This Lecture

Linear Regression

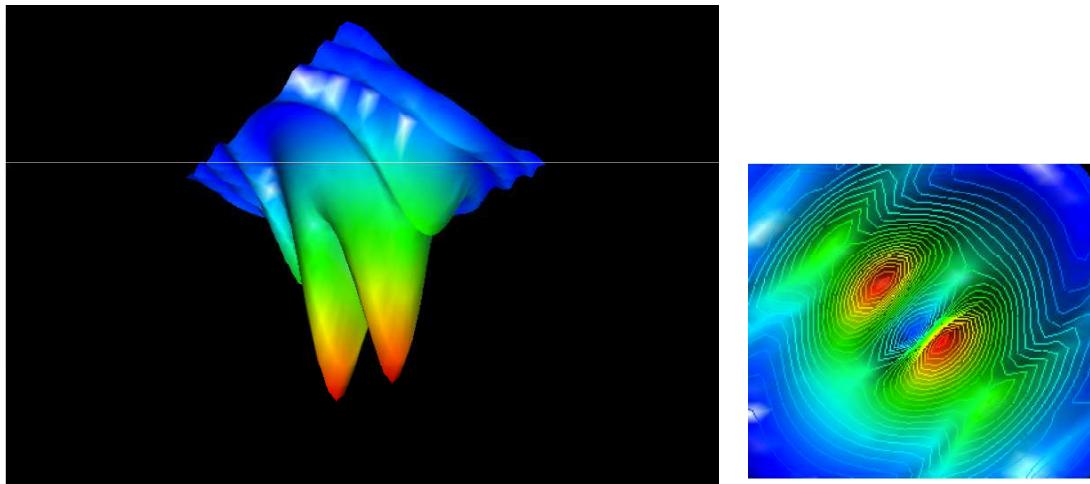
Error (a.k.a. objective, cost, loss) function

→ Gradient descent (GD) basics

# Gradient Descent Intuition

$$\underset{\mathbf{w}}{\operatorname{argmin}} E(\mathbf{w})$$

- Walking downhill



J. Braun

31

## Gradient Descent

- A. Cauchy, 1847
- “Méthode générale pour la résolution des systèmes d'équations simultanées”

in

*Compte rendu de séances de l'académie des sciences, pg. 536-538*

( 536 )

équation, était non plus la distance  $r$  ou  $\nu$ , mais l'une des coordonnées rectangulaires de l'astre observé. »

ANALYSE MATHÉMATIQUE. — *Méthode générale pour la résolution des systèmes d'équations simultanées*; par M. AUGUSTIN CAUCHY.

« Étant donné un système d'équations simultanées qu'il s'agit de résoudre, on commence ordinairement par les réduire à une seule, à l'aide d'éliminations successives, sauf à résoudre définitivement, s'il se peut, l'équation résultante. Mais il importe d'observer, 1<sup>o</sup> que, dans un grand nombre de cas, l'élimination ne peut s'effectuer en aucune manière; 2<sup>o</sup> que l'équation résultante est généralement très-compliquée, lors même que les équations données sont assez simples. Pour ces deux motifs, on conçoit qu'il serait très-utile de connaître une méthode générale qui puisse servir à résoudre directement un système d'équations simultanées. Telle est celle que j'ai obtenue, et dont je vais dire ici quelques mots. Je me bornerai pour l'instant à indiquer les principes sur lesquels elle se fonde, me proposant de revenir avec plus de détails sur le même sujet, dans un prochain Mémoire.

« Soit d'abord

$$u = f(x, y, z)$$

une fonction de plusieurs variables  $x, y, z, \dots$ , qui ne devienne jamais négative et qui reste continue, du moins entre certaines limites. Pour trouver les valeurs de  $x, y, z, \dots$ , qui vérifieront l'équation

$$(1) \quad u = 0,$$

il suffira de faire décroître indéfiniment la fonction  $u$ , jusqu'à ce qu'elle s'évanouisse. Or soient

$$x, y, z, \dots$$

des valeurs particulières attribuées aux variables  $x, y, z, \dots$ ;  $u$  la valeur correspondante de  $u$ ;  $X, Y, Z, \dots$  les valeurs correspondantes de  $D_x u, D_y u, D_z u, \dots$ , et  $\alpha, \beta, \gamma, \dots$  des accroissements très-petits attribués aux valeurs particulières  $x, y, z, \dots$ . Quand on posera

$$x = x + \alpha, \quad y = y + \beta, \quad z = z + \gamma, \dots$$

on aura sensiblement

$$(2) \quad u = f(x + \alpha, y + \beta, \dots) = u + \alpha X + \beta Y + \gamma Z + \dots$$

# Gradient Descent (GD) Algorithm

$$\underset{w_0, w_1, \dots, w_n}{\operatorname{argmin}} E(w_0, w_1, w_2, \dots, w_n)$$

$$\underset{\mathbf{w}}{\operatorname{argmin}} E(\mathbf{w})$$

Initialize  $w$

Set initial values of model parameters

Repeat {

$$w := w - \alpha \frac{\partial E(w)}{\partial w}$$

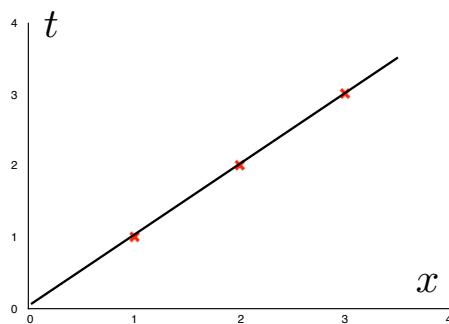
Simultaneously for all components  $w_j$  of vector  $w$

} until converged

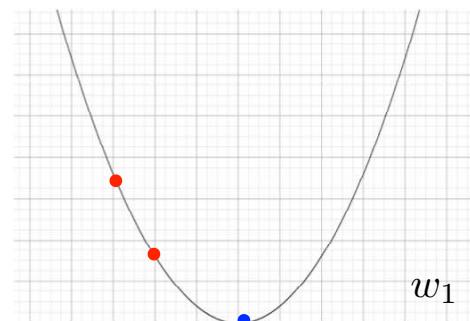
# Gradient Descent for LSE

$$h_w(x) = w_0 + w_1 x$$

$$E(w_0, w_1) = \frac{1}{2N} \sum_{i=1}^N (h_w(x_i) - t_i)^2$$



$$E(w_1)$$



Repeat {

$$w := w - \alpha \frac{\partial E(w)}{\partial w}$$

} until converged

# Gradient Descent for LSE

$$E(w_0, w_1) = \frac{1}{2N} \sum_{i=1}^N (h_w(x_i) - t_i)^2$$

- For single exemplar:

$$\begin{aligned}\frac{\partial E(w)}{\partial w_j} &= \frac{\partial}{\partial w_j} \frac{1}{2} (h_w(x) - t)^2 = \\ &= 2 \frac{1}{2} (h_w(x) - t) \frac{\partial}{\partial w_j} (h_w(x) - t) = \\ &= (h_w(x) - t) \frac{\partial}{\partial w_j} \left( \sum_{i=0}^n (w_i x_i - t) \right) = \\ &= (h_w(x) - t) x_j\end{aligned}$$

# Gradient Descent (GD) Algorithm

$$\underset{w_0, w_1, \dots, w_n}{\operatorname{argmin}} E(w_0, w_1, w_2, \dots, w_n) \quad \underset{\mathbf{w}}{\operatorname{argmin}} E(\mathbf{w})$$

Initialize  $w$

Set initial values of model parameters

Repeat {

$$w_j := w_j + \alpha \sum_{i=0}^N (t^{(i)} - h_w(x^{(i)})) x_j^{(i)}$$

Simultaneously for all components  $w_j$  of vector  $w$

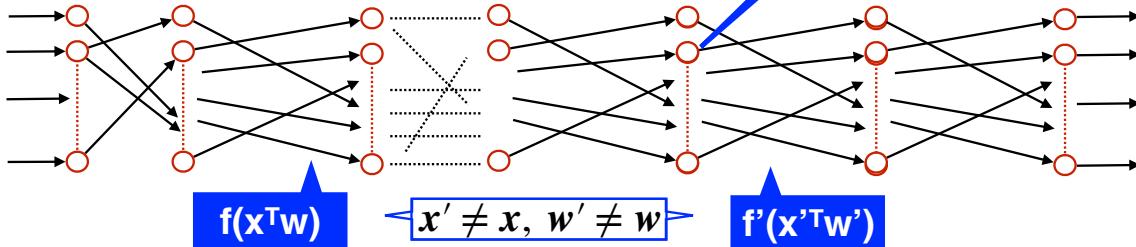
} until converged

- For linear regression, closed-form alternative to this iterative approach exists
  - Referred to as *normal equations*
- But, as we will see, in artificial neural networks gradient-descent will be done using iterative procedures

# Linear Constructs in ANNs

Linear constructs (such as those we see in linear regression) appear in neural networks (ANNs)

$$f(\mathbf{x}, \mathbf{w}) = x_1 w_1 + \cdots + x_n w_n \quad f(\mathbf{w}^T \mathbf{x})$$



Note *linear constructs* as “elements” inside ANN

Hence, need to understand  
(study) linear models

Although, as we will see,  
ANNs are NOT linear models!

## Questions?

---

# **Neural Networks and Deep Learning**

**Dr. Jerome J. Braun**

## **This Lecture: Supervised-Learning Basics II**

Course: Neural Networks and Deep Learning  
IE 7615

---

---

**Unauthorized distributing, redistributing, posting,  
and/or reposting, of the materials of this course  
(including course lectures, lecture slides, slide-sets,  
syllabi, guidelines, assignments, quizzes, exams,  
presentations, software, electronic media, etc.),  
is prohibited.**

# This Lecture

---



## Maximum Likelihood

Modeling data uncertainty (linear regression context)

Maximum-likelihood solution for linear regression

## Maximum Likelihood – Example

---

- Coin tossing
  - H — head
  - T — tail
- Model
  - Bernoulli random variable X
  - X can be
    - 1 for Head
    - 0 for Tail
  - $p(X=1) = w$
  - $p(X=0) = 1 - w$



$$X_1 = 0 \quad X_1 = 1$$

Parameter w to be  
determined from data

## Maximum Likelihood – Example

- Coin tossing
  - H – head
  - T – tail



$X_1 = 1$

$X_2 = 0$

$X_3 = 1$

$X_4 = 1$

$X_5 = 0$

$$p(D|w) = p(X_1, \dots, X_5) = w^3(1-w)^2$$

Maximum Likelihood – w chosen so as to maximize likelihood

J. Braun

5

## Maximum Likelihood – Example

- Coin tossing
  - H – head
  - T – tail



$X_1 = 1$

$X_2 = 0$

$X_3 = 1$

$X_4 = 1$

$X_5 = 0$

$$p(D|w) = p(X_1, \dots, X_5) = w^3(1-w)^2$$

(#heads divided by  
total #tosses)

$$w_{ML} = \frac{3}{3+2}$$

Show this –  
left as exercise

Ex.

J. Braun

6

## Maximum Likelihood

- More generally:

$$X \ p(X|w)$$

- N observations i.i.d.

$$D = \{x_1, \dots, x_N\}$$

$$\text{Likelihood}(D) = \prod_{i=1}^N p(x_i|w)$$

$$w_{ML} = \underset{w}{\operatorname{argmax}} (\text{Likelihood}(D)) =$$

$$= \underset{w}{\operatorname{argmax}} \left( \sum_{i=1}^N \log p(X_i|w) \right)$$

Log Likelihood

J. Braun

7

## Maximum Likelihood

Set of m exemplars  $\mathbb{X} = \{x^{(1)}, \dots, x^{(m)}\}$

Drawn independently from TRUE but UNKNOWN  $p_{\text{data}}(\mathbf{x})$

Denote by

$$p_{\text{model}}(\mathbf{x}; \boldsymbol{\theta})$$

- Parametric family of distributions
- Parametrized by  $\boldsymbol{\theta}$

Maximum likelihood estimator is defined as

$$\boldsymbol{\theta}_{\text{ML}} = \arg \max_{\boldsymbol{\theta}} p_{\text{model}}(\mathbb{X}; \boldsymbol{\theta})$$

$$= \arg \max_{\boldsymbol{\theta}} \prod_{i=1}^m p_{\text{model}}(\mathbf{x}^{(i)}; \boldsymbol{\theta})$$

J. Braun

8

# Maximum Likelihood

$$\theta_{\text{ML}} = \arg \max_{\theta} \prod_{i=1}^m p_{\text{model}}(\mathbf{x}^{(i)}; \boldsymbol{\theta})$$

Prone to underflows

$$\theta_{\text{ML}} = \arg \max_{\theta} \sum_{i=1}^m \log p_{\text{model}}(\mathbf{x}^{(i)}; \boldsymbol{\theta})$$

Alleviate by transitioning  
to log domain

Denote by

$$\hat{p}_{\text{data}}$$

- Empirical distribution
- Defined by TRAINING data

- Rescaling  $\theta_{\text{ML}}$  (above) by  $1/m$
- Yields expectation w.r.t. that empirical distribution
- (i.e., distribution defined by training data)

$$\theta_{\text{ML}} = \arg \max_{\theta} \mathbb{E}_{\mathbf{x} \sim \hat{p}_{\text{data}}} \log p_{\text{model}}(\mathbf{x}; \boldsymbol{\theta})$$

J. Braun

9

# Conditional Log Likelihood

$$\theta_{\text{ML}} = \arg \max_{\theta} \mathbb{E}_{\mathbf{x} \sim \hat{p}_{\text{data}}} \log p_{\text{model}}(\mathbf{x}; \boldsymbol{\theta})$$

Goal: estimate conditional probability  $P(\mathbf{y} \mid \mathbf{x}; \boldsymbol{\theta})$

Typical for supervised learning

Predict y ...

... given x

All targets

All inputs

$$\theta_{\text{ML}} = \arg \max_{\theta} P(\mathbf{Y} \mid \mathbf{X}; \boldsymbol{\theta})$$

Assuming i.i.d.  
training exemplars

$$\theta_{\text{ML}} = \arg \max_{\theta} \sum_{i=1}^m \log P(\mathbf{y}^{(i)} \mid \mathbf{x}^{(i)}; \boldsymbol{\theta})$$

J. Braun

10

# This Lecture

---

**Maximum Likelihood**

- **Modeling data uncertainty (linear regression context)**
- Maximum-likelihood solution for linear regression**

## Noisy Observations

---

**True target values are unobservable**

$\underline{t}$

**Observed target values burdened by noise**

$\epsilon$

**Only noisy “versions” of target values are observed**

$t$

**Noisy target-value**

**True value — unobservable**

$$t = \underline{t} + \epsilon \quad \text{Noise}$$

**Assume Gaussian noise distribution**

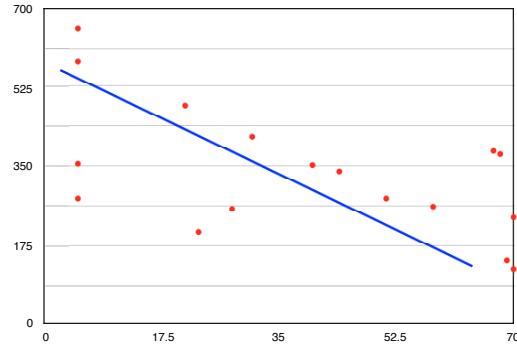
$$N(\epsilon | 0, \beta^{-1})$$

$$\text{Precision} \quad \beta = \frac{1}{\sigma^2}$$

**Variance**

# Modeling Data Uncertainty

$$h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$$



- Parameters  $\mathbf{w}$
- Data  $D = \{\mathbf{x}, t\}$ 
  - For  $N$  exemplars
- Objective:  $\text{argmax } P(D \mid \text{model})$ 
  - Maximize probability of data  $D$  given model

## IID Assumption for $t$

N inputs

$$\mathbf{x} = (x_1, \dots, x_N)^T$$

N target values

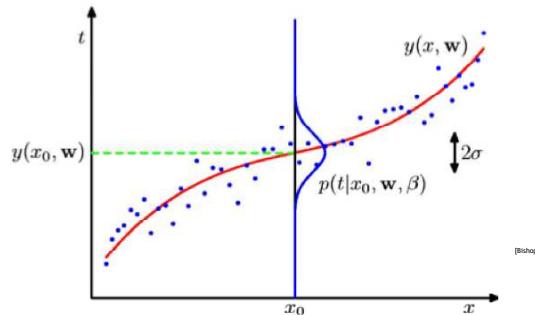
$$\mathbf{t} = (t_1, \dots, t_N)^T$$

Assume  $t_i$  are IID (i.i.d. — independent, identically distributed)

# Minimizing Expected Squared Loss

- Now use training data  $\{x, t\}$  to determine unknown parameters  $w$  and  $\beta$ 
  - By maximum likelihood

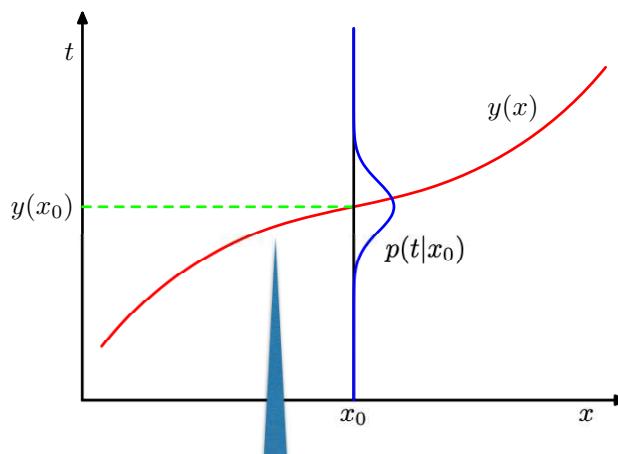
$$p(t|x, w, \beta) = \mathcal{N}(t|y(x, w), \beta^{-1})$$



If assume data drawn independently from above distribution, then

$$p(\mathbf{t}|\mathbf{x}, w, \beta) = \prod_{n=1}^N \mathcal{N}(t_n|y(x_n, w), \beta^{-1})$$

# Minimizing Expected Squared Loss



Regression function  $y(x)$  which minimizes expected squared loss, is given by mean of conditional distribution  $p(t|x)$

# Maximum Likelihood Solution

---

$$p(t|x, \mathbf{w}, \beta) = \mathcal{N}(t|y(x, \mathbf{w}), \beta^{-1})$$

$$\mathbf{w}_{ML} = \operatorname{argmax}_{\mathbf{w}} p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta)$$

$$\beta_{ML} = \operatorname{argmax}_{\beta} p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta)$$

# Maximizing Likelihood

---

Goal — maximize likelihood

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \prod_{n=1}^N \mathcal{N}(t_n|y(x_n, \mathbf{w}), \beta^{-1})$$

More convenient to maximize LOG LIKELIHOOD

$$\ln p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = -\frac{\beta}{2} \sum_{n=1}^N \{y(x_n, \mathbf{w}) - t_n\}^2 + \frac{N}{2} \ln \beta - \frac{N}{2} \ln(2\pi).$$

Q: What can be said about each of these terms in context of maximization?

# Maximizing Likelihood and SSE

Maximizing log-likelihood

$$\ln p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = -\frac{\beta}{2} \sum_{n=1}^N \{y(x_n, \mathbf{w}) - t_n\}^2 + \frac{N}{2} \ln \beta - \frac{N}{2} \ln(2\pi).$$

Multiply by and  
change argmax to  
argmin

$$-\frac{1}{N\beta}$$

These can be omitted from  
optimization — since they  
do not depend on  $\mathbf{w}$

Same as minimizing  
SUM-OF-SQUARES ERROR

$$\frac{1}{2} \sum_{n=1}^N \{y(x_n, \mathbf{w}) - t_n\}^2$$

## Max Likelihood — Probabilistic Motivation for SSE

$$\mathbf{w}_{ML} = \underset{\mathbf{w}}{\operatorname{argmax}} p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{2} \sum_{n=1}^N \{y(x_n, \mathbf{w}) - t_n\}^2$$

$$\beta_{ML} = \underset{\beta}{\operatorname{argmax}} p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) \quad \text{Ex. } \leftarrow \text{(left as exercise)}$$

- Under assumption of Gaussian noise distribution
  - Maximizing likelihood is equivalent to minimizing SSE
  - Motivates use of SSE for linear regression
- Minimizing SSE — “Least Squares method”

# Frequentist vs. Bayesian Approaches

- Frequentist viewpoint

- $w$  viewed as fixed parameter
- $w$  determined by some estimator
  - Error bars on estimate determined by considering distribution of possible data sets  $D$
  - Maximum Likelihood — frequentist estimator

$$\mathcal{D} = \{t_1, \dots, t_N\}$$

$$p(\mathcal{D}|w)$$

- Bayesian viewpoint

- Single data set  $D$  — the one actually observed
- Uncertainty in  $w$  expressed through probability distribution over  $w$

max likelihood

max posterior →  $p(w|\mathcal{D}) = \frac{p(\mathcal{D}|w)p(w)}{p(\mathcal{D})}$

Questions?

---

# **Neural Networks and Deep Learning**

**Dr. Jerome J. Braun**

## **This Lecture: Supervised-Learning Basics III**

Course: Neural Networks and Deep Learning  
IE 7615

---

---

**Unauthorized distributing, redistributing, posting,  
and/or reposting, of the materials of this course  
(including course lectures, lecture slides, slide-sets,  
syllabi, guidelines, assignments, quizzes, exams,  
presentations, software, electronic media, etc.),  
is prohibited.**

# This Lecture

---



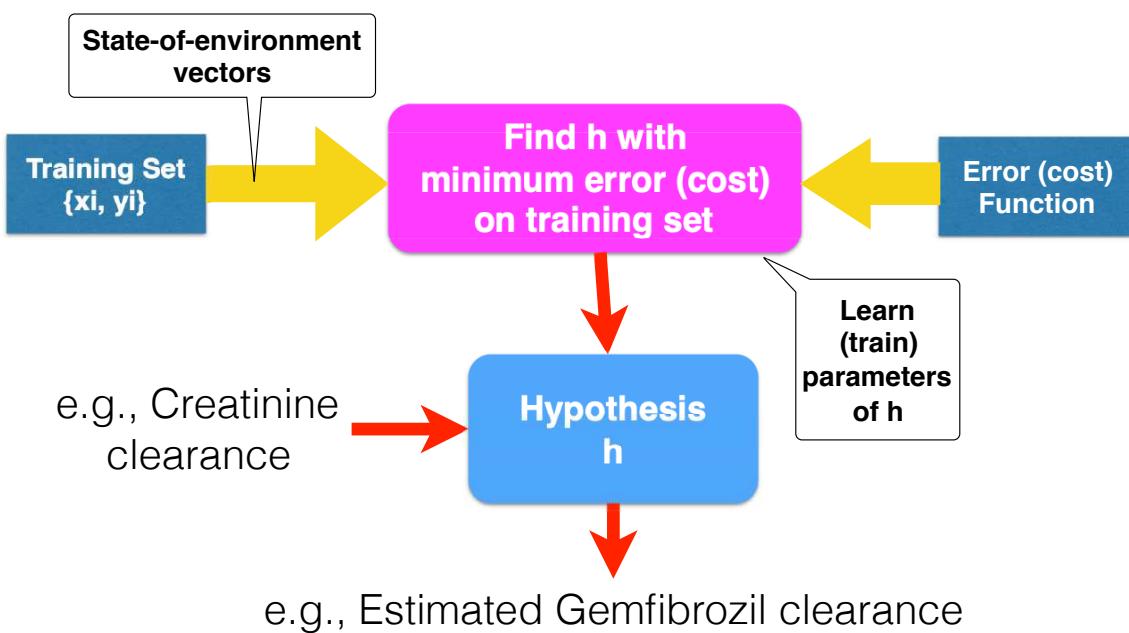
Recap — Linear regression

Linear regression with nonlinear features

Linear regression — closed-form solution

## Recap: Learning the Hypothesis Function

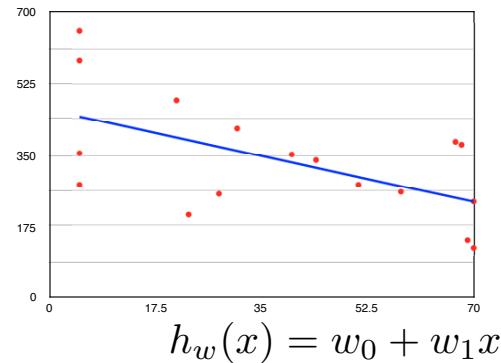
---



## Recap: Linear Regression Maximum Likelihood Solution

$$h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$$

- Under assumption of Gaussian noise distribution
  - Maximizing likelihood is equivalent to minimizing SSE
  - Motivates use of SSE for linear regression
- Minimizing SSE — “Least Squares method”
- For linear hypotheses — closed-form solution
  - Normal equations (we will see this shortly)



$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \prod_{n=1}^N \mathcal{N}(t_n | y(x_n, \mathbf{w}), \beta^{-1})$$

$$\mathbf{w}_{ML} = \underset{\mathbf{w}}{\operatorname{argmax}} p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{2} \sum_{n=1}^N \{y(x_n, \mathbf{w}) - t_n\}^2$$

## This Lecture

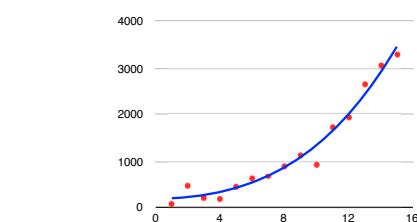
Recap — Linear regression



Linear regression with nonlinear features

Linear regression — closed-form solution

# Nonlinear Combinations of Input Variables



$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x})$$

Basis functions—functions of inputs

$$y(x, w) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + \dots$$

Examples:

e.g., `creat_clear`

$$y(x, w) = w_0 + w_1(\text{input}) + w_2(\text{input})^2 + w_3(\text{input})^3 + \dots$$

$$y(x, w) = w_0 + w_1(\text{input}) + w_2 \sqrt{(\text{input})} + w_3(\text{input})^{\frac{1}{3}} + \dots$$

- Function  $y(\mathbf{x}, \mathbf{w})$  — nonlinear function of input vector  $\mathbf{x}$
- Linear in new feature space (features — functions of input)
- Linear models — because  $y(\mathbf{x}, \mathbf{w})$  is linear in parameters  $\mathbf{w}$
- Simplifies model analysis
- Implies significant limitations

# Basis Functions

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x})$$

- E.g.,
- Gaussian
- Sigmoidal
- Fourier
- other....

$$y(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$$

$$\phi_j(x) = \exp \left\{ -\frac{(x - \mu_j)^2}{2s^2} \right\}$$

$$\phi_j(x) = \sigma \left( \frac{x - \mu_j}{s} \right)$$

$$\sigma(a) = \frac{1}{1 + \exp(-a)}.$$

# This Lecture

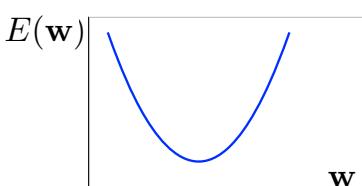
Recap — Linear regression

Linear regression with nonlinear features



Linear regression — closed-form solution

## Closed-Form Minimization



$$E(w_0, w_1, w_2, \dots, w_n) = \frac{1}{2N} \sum_{i=1}^N (h_w(x_i) - t_i)^2$$

Need to find  $w$  that minimizes  $E$

$$E(w) = \frac{1}{2} \sum_{n=1}^N \{t_n - w^T \phi(x_n)\}^2$$

Quadratic function of  $w$

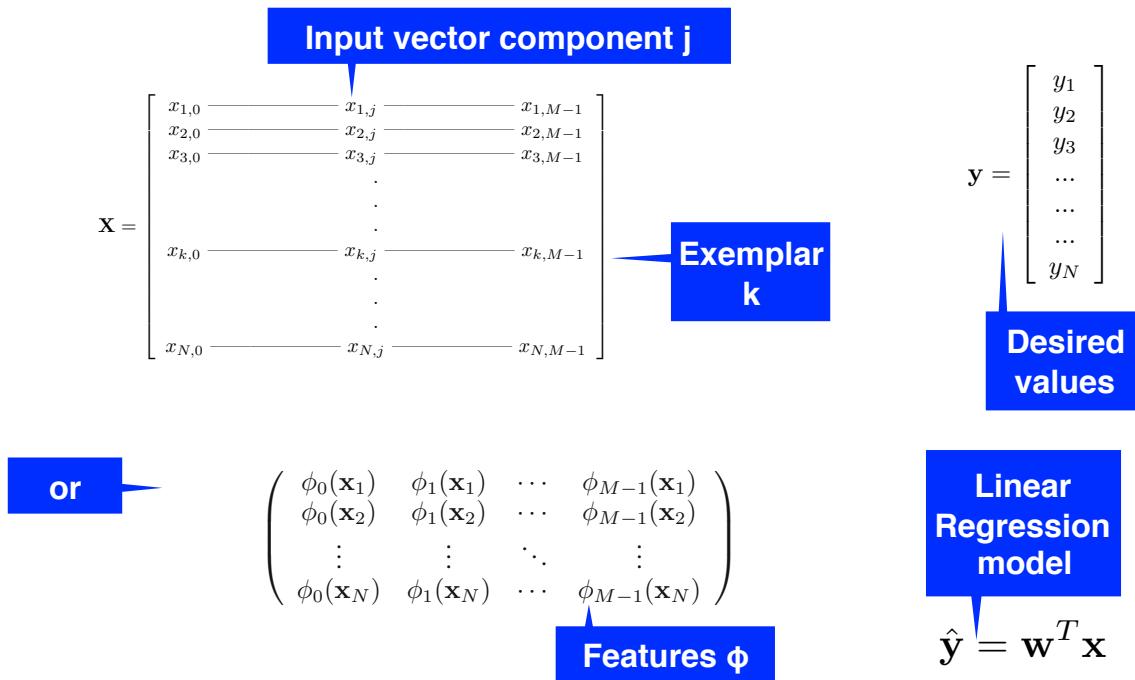
$$\frac{\partial E(w)}{\partial w_j} = 0$$

For every  $j$

- Since  $E$  is quadratic function of  $w$ 
  - Minimizer can be found in closed form

Closed-form solution does NOT exist for neural networks

# Design Matrix



J. Braun

11

# Normal Equations

$$\nabla_{\mathbf{w}} \text{MSE}_{\text{train}} = 0$$

$\mathbf{X}$  = design matrix

$$\nabla_{\mathbf{w}} \frac{1}{N} \|\hat{\mathbf{y}}^{(\text{train})} - \mathbf{y}^{(\text{train})}\|_2^2 = 0$$

Linear Regression  
model

$$\frac{1}{N} \nabla_{\mathbf{w}} \|\mathbf{X}^{(\text{train})} \mathbf{w} - \mathbf{y}^{(\text{train})}\|_2^2 = 0$$

$$\hat{\mathbf{y}} = \mathbf{w}^T \mathbf{x}$$

**EXERCISE:** Derive expression for  $\mathbf{w}$  from the above (final result shown in next slide)

$$\mathbf{w} =$$

Hints:  
Expand II...II, combine/simplify terms, and take gradient.

$$\nabla_{\mathbf{x}} \mathbf{y} \leftarrow \text{Gradient of } \mathbf{y} \text{ w.r.t. } \mathbf{x}$$

$$\|\mathbf{x}\|_2 = \sqrt{\sum_i |x_i|^2}$$

$$\|\cdot\|^2 = (\|\mathbf{x}\|)^2$$

# Normal Equations

$$\nabla_{\mathbf{w}} \text{MSE}_{\text{train}} = 0$$

$$\nabla_{\mathbf{w}} \frac{1}{N} \|\hat{\mathbf{y}}^{(\text{train})} - \mathbf{y}^{(\text{train})}\|_2^2 = 0$$

$$\frac{1}{N} \nabla_{\mathbf{w}} \|\mathbf{X}^{(\text{train})} \mathbf{w} - \mathbf{y}^{(\text{train})}\|_2^2 = 0$$

Linear  
Regression  
model

$$\hat{\mathbf{y}} = \mathbf{w}^T \mathbf{x}$$

Hence

$$\mathbf{w} = (\mathbf{X}^{(\text{train})\top} \mathbf{X}^{(\text{train})})^{-1} \mathbf{X}^{(\text{train})\top} \mathbf{y}^{(\text{train})}$$

Normal  
Equations

or

$$\Theta = (X^T X)^{-1} X^T y$$

## Normal Equations – Example

Inputs/features (1 to n)

x0	x1	x2	x3	x4	y
...	...	...	...	...	...
1	41	1	20	19	12
1	51	2	20	19	12
1	22	2	12	18	2
1	20	2	13	18	2
1	53	1	14	18	3
1	29	2	16	18	4
...	...	...	...	...	...
...	...	...	...	...	...

$$X = \begin{bmatrix} 1 & 41 & 1 & 20 & 19 \\ 1 & 51 & 2 & 20 & 19 \\ 1 & 22 & 2 & 12 & 18 \\ 1 & 20 & 2 & 13 & 18 \\ 1 & 53 & 1 & 14 & 18 \\ 1 & 29 & 2 & 16 & 18 \end{bmatrix} \quad y = \begin{bmatrix} 12 \\ 12 \\ 2 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

$$\Theta = (X^T X)^{-1} X^T y$$

## Normal Equations vs. Gradient Descent (For Linear Regression)

	GRADIENT DESCENT	NORMAL EQUATIONS
Need to select Learning Rate	YES	NO
Need to iterate	YES	NO
Need for matrix inversion	NO	YES
Speed issue for large number of features	NO	YES

Closed-form solution does NOT exist for neural networks

Questions?

---

# **Neural Networks and Deep Learning**

**Dr. Jerome J. Braun**

## **This Lecture: Supervised-Learning Basics IV**

Course: Neural Networks and Deep Learning  
IE 7615

---

---

**Unauthorized distributing, redistributing, posting,  
and/or reposting, of the materials of this course  
(including course lectures, lecture slides, slide-sets,  
syllabi, guidelines, assignments, quizzes, exams,  
presentations, software, electronic media, etc.),  
is prohibited.**

# This Lecture



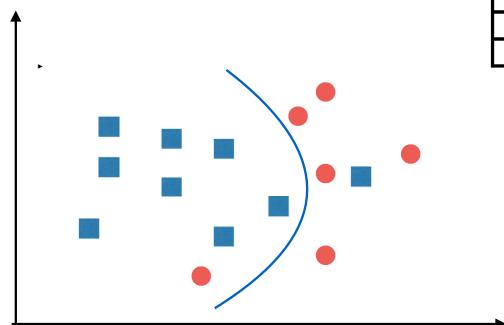
Logistic regression

Math supplement: matrix differentiation

## Classification (a.k.a. Recognition, Categorization)

### • Examples:

- Lesion — benign or malignant
- Audio signal — speech or music
- Transaction — legitimate or fraudulent
- Cell — normal or abnormal (e.g., mutated)
- Genomic sequence ...
- ... many more ...



A few examples (out of many)	
Machine Vision	Speech & Language Processing (NLP)
Face Recognition, Object Recognition	Autonomous Vehicles
Self-driving cars	Robotics
Forecasting	Assistive Robotics
Weather, climate, etc.	Bioinformatics
Computational biology, neuroscience	"X"-informatics
Analytics (text, video, etc.)	Economics, Financial forecasting
Medical diagnostics	Insurance
Biomed data analysis; healthcare	Fraud detection
Drug development	Image/photo tagging
Personalized medicine	
... many others...	

$$\begin{aligned}y \in \{0, 1\} &\quad y \in \{\text{NO, YES}\} \\y \in \{A, B\} &\quad y \in \{\text{Class1, Class2}\}\end{aligned}$$

Two-class

$$y \in \{1, 2, 3, 4, \dots\}$$

K-class

# Least Squares for Classification

- Least squares error function for classification?

- Possible

- E.g., for n-th exemplar,  $y_n(k)=1$  if class k, else 0

- Apply LSE as for regression

- Linear regression for K-class problems

- Each class  $C_k$  described by its own linear model

- [See Bishop 4.1.3]

$$y_n \in \{0, 1\}$$

For two class problem

$$y_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + w_{k0}$$

- LSE approach problematic (next slides)

- Less robust than logistic regression (next)

- Sometimes leads to poor results

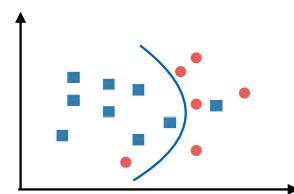
J. Braun

# Logistic Regression

- Classification by Logistic Regression

- Misnomer

- Logistic *regression* is not regression

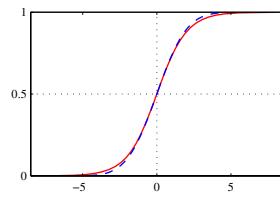


- Consider 2-class problem

- $h(\mathbf{x})$  should be 0 or 1

$$g(a) = \frac{1}{1 + \exp(-a)}$$

Sigmoid function



(0,1)

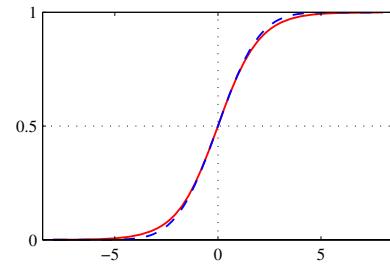
Any value of  $h_w(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$

J. Braun

# Logistic Sigmoid and Logistic Regression

$$0 \leq h_{\theta}(x) \leq 1$$

$$h_{\theta}(x) = g(\theta^T x) = \frac{1}{1 + \exp(-\theta^T x)}$$



$$g(a) = \frac{1}{1 + \exp(-a)}$$

$$p(y = 1|x) = \frac{1}{1 + \exp(-\theta^T x)}$$

Prediction

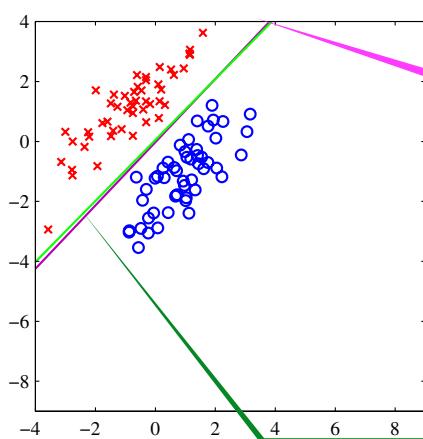
y=0 if  $h_{\theta}(x) < 0.5$

y=1 if  $h_{\theta}(x) \geq 0.5$

J. Braun

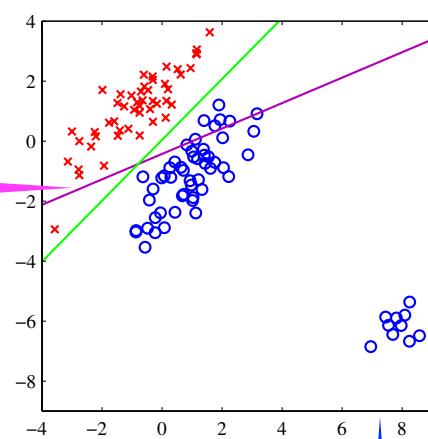
7

## Least Squares vs. Logistic Regression for Classification Least Squares Non-robustness



By Least Squares

By Logistic Regression



Results when extra data points added (at bottom left)

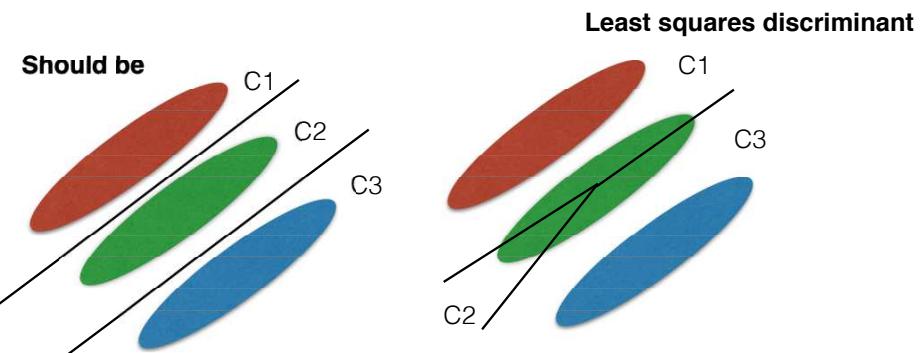
Least squares highly sensitive to outliers, unlike logistic regression

J. Braun

8

## Issues with Least Squares for Classification

- Non-robustness
- Poor results on simple multi-class problems
  - Example (see Bishop 4.1.3):



J. Braun

9

## Logistic Regression Error Function

Hypothesis  $\rightarrow h_{\theta}(x) = g(\theta^T x) = \frac{1}{1 + \exp(-\theta^T x)}$

Parameters  $\rightarrow D = \{x^{(i)}, y^{(i)}\}$

Data  $\rightarrow$

- Error (loss, cost) function:  
$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m \left\{ y^{(i)} \ln h_{\theta}(x^{(i)}) + (1 - y^{(i)}) \ln (1 - h_{\theta}(x^{(i)})) \right\}$$

This kind of loss function is referred to as a cross-entropy loss function

We will justify why we refer to the above as a “cross-entropy” in next lecture, in which we will discuss selected information-theory topics relevant to deep learning

- Objective: 
$$\operatorname{argmin}_{\theta} J(\theta)$$

J. Braun

10

## Maximum Likelihood for Logistic Regression

$$\begin{aligned} J(\theta) &= -\frac{1}{m} \sum_{i=1}^m \text{Cost}\left(h_{\theta}(x^{(i)}, y^{(i)})\right) = \\ &= -\frac{1}{m} \sum_{i=1}^m \left\{ y^{(i)} \ln h_{\theta}(x^{(i)}) + (1 - y^{(i)}) \ln (1 - h_{\theta}(x^{(i)})) \right\} \end{aligned}$$

- Cost (error) for Logistic Regression can be derived using maximum likelihood

Ex.

- Compute derivative w.r.t. parameters  $\theta$

## Maximum Likelihood for Logistic Regression

- No closed-form solution for Logistic Regression

$$\begin{aligned} J(\theta) &= -\frac{1}{m} \sum_{i=1}^m \text{Cost}\left(h_{\theta}(x^{(i)}, y^{(i)})\right) = \\ &= -\frac{1}{m} \sum_{i=1}^m \left\{ y^{(i)} \ln h_{\theta}(x^{(i)}) + (1 - y^{(i)}) \ln (1 - h_{\theta}(x^{(i)})) \right\} \end{aligned}$$

- Gradient Descent  $\underset{\theta}{\operatorname{argmin}} J(\theta)$

Repeat {

$$\theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$$

Simultaneously  
for all  $\theta_j$

}

## Maximum Likelihood for Logistic Regression

- No closed-form solution for Logistic Regression

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m \text{Cost}\left(h_\theta(x^{(i)}, y^{(i)})\right) = \\ = -\frac{1}{m} \sum_{i=1}^m \left\{ y^{(i)} \ln h_\theta(x^{(i)}) + (1 - y^{(i)}) \ln (1 - h_\theta(x^{(i)})) \right\}$$

- Gradient Descent  $\underset{\theta}{\operatorname{argmin}} J(\theta)$   
Repeat {

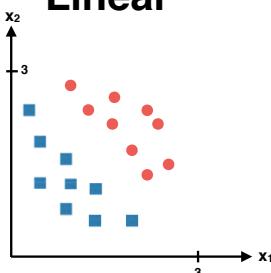
$$\theta_j := \theta_j - \alpha \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Simultaneously  
for all  $\theta_j$

}

## Decision Boundaries

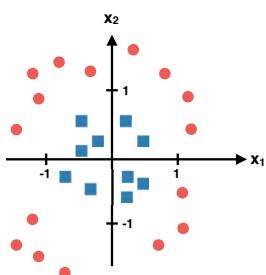
### Linear



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

Predict  $y=1$  if:  $-3 + x_1 + x_2 \geq 0$

### Nonlinear



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

Predict  $y=1$  if:  $-1 + x_1^2 + x_2^2 \geq 0$

# This Lecture

---

Logistic regression



Math supplement: matrix differentiation

# Matrix Differentiation

---

- Let

$$\mathbf{y} = \psi(\mathbf{x})$$

- where  $\mathbf{y}$  is m-dim vector,  $\mathbf{x}$  is n-dim vector
- Matrix of  $m \times n$  matrix of partial derivatives of transformation from  $\mathbf{x}$  to  $\mathbf{y}$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \dots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \dots & \frac{\partial y_2}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \dots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

← Jacobian matrix

# Matrix Differentiation

---

- If  $y = Ax$
- where
  - ♦  $y$  is  $m \times 1$ , and
  - ♦  $x$  is  $n \times 1$ , and
  - ♦  $A$  is  $m \times n$ , AND  $A$  is independent of  $x$

- Then

$$\frac{\partial y}{\partial x} = A$$

---

J. Braun

# Matrix Differentiation

---

- If  $\alpha = y^T Ax$
- where
  - ♦  $\alpha$  is scalar
  - ♦  $y$  is  $m \times 1$ , and
  - ♦  $x$  is  $n \times 1$ , and
  - ♦  $A$  is  $m \times n$ , AND  $A$  is independent of  $x$  AND of  $y$

- Then

$$\frac{\partial \alpha}{\partial x} = y^T A \quad \text{and} \quad \frac{\partial \alpha}{\partial y} = x^T A^T$$

---

J. Braun

# Matrix Differentiation

- If

$$\alpha = \mathbf{x}^T \mathbf{A} \mathbf{x}$$

Scalar  $\alpha$  given by quadratic form

- where

•  $\alpha$  is scalar, given by quadratic form

•  $\mathbf{x}$  is  $n \times 1$ ,  $\mathbf{A}$  is  $n \times n$ , AND  $\mathbf{A}$  does not depend on  $\mathbf{x}$

- Then

$$\frac{\partial \alpha}{\partial \mathbf{x}} = \mathbf{x}^T (\mathbf{A} + \mathbf{A}^T)$$

- If  $\mathbf{A}$  is symmetric (special case), then

$$\frac{\partial \alpha}{\partial \mathbf{x}} = 2\mathbf{x}^T \mathbf{A}$$

J. Braun

Questions?

---

# **Neural Networks and Deep Learning**

**Dr. Jerome J. Braun**

## **This Lecture: Information-Theory Basics for Deep Learning**

Course: Neural Networks and Deep Learning  
IE 7615

---

---

**Unauthorized distributing, redistributing, posting,  
and/or reposting, of the materials of this course  
(including course lectures, lecture slides, slide-sets,  
syllabi, guidelines, assignments, quizzes, exams,  
presentations, software, electronic media, etc.),  
is prohibited.**

# Measure of Information Content

- Measure of surprise (information) of observing  $X=x$
- Suppose we are told that
  - › Highly improbable event has just occurred
    - ♦ More information
  - › Likely event has just occurred
    - ♦ Less information
  - › Event that was certain to occur has just occurred
    - ♦ No information
- Measure of information content must therefore depend on probability distribution  $p(x)$ 
  - › Seek quantity  $h(x)$  that is monotonic function of probability  $p(x)$

J. Braun

3

## Form of Information-Content Measure

- Measure  $h(x)$  to be monotonic function of probability  $p(x)$
- Form of  $h(\cdot)$ 
  - › If observing BOTH of two unrelated events  $x$  and  $y$ 
    - ♦ Information gain should be sum of information gained from each separately, i.e.,
      - $h(x, y) = h(x) + h(y)$
  - › Two unrelated events statistically independent, i.e.,
    - $p(x, y) = p(x)p(y)$
  - › From above can show that  $h(x)$  must be given by log function of  $p(x)$

$$h(x) = -\log_2 p(x)$$

Minus sign  
ensure  $h$  is  
positive or zero

J. Braun

4

# Shannon Entropy

- Suppose sender wants to transmit value of random variable to receiver
  - Average amount of information transmitted is:
    - ♦ Expectation of  $h$  w.r.t. distribution  $p(x)$

Entropy →

$$H[x] = - \sum_x p(x) \log_2 p(x)$$

Since

$$\lim_{p \rightarrow 0} p \ln p = 0$$

For any  $x$  s.t.  $p(x)=0$

$$p(x) \ln p(x) = 0$$

# Shannon Entropy

Self-information  
of event  $x=x$

$$h(x) = -\log_2 p(x)$$

or

$$I(x) = -\log P(x)$$

Deals  
with  
**SINGLE**  
outcome

Units:  
Log base 2: bits  
Log base e: nats

How can quantify uncertainty in ENTIRE probability distribution ?

Shannon entropy

$$H(x) = \mathbb{E}_{x \sim P}[I(x)] = -\mathbb{E}_{x \sim P}[\log P(x)]$$

Expected amount of information in event drawn from distribution  $P$

Lower bound on number of units (bits if  $\log_2$ ) needed ON AVERAGE to encode symbols drawn from distribution  $P$

# Entropy Examples

Random variable  $x$  that has eight equiprobable states

$$H[x] = -8 \times \frac{1}{8} \log_2 \frac{1}{8} = 3 \text{ bits}$$

Random variable  $x$  that has eight possible states  $\{a, b, c, d, e, f, g, h\}$  with respective probabilities  $\{1/2, 1/4, 1/8, 1/16, 1/64, 1/64, 1/64, 1/64\}$

$$H[x] = -\frac{1}{2} \log_2 \frac{1}{2} - \frac{1}{4} \log_2 \frac{1}{4} - \frac{1}{8} \log_2 \frac{1}{8} - \frac{1}{16} \log_2 \frac{1}{16} - \frac{4}{64} \log_2 \frac{1}{64} = 2 \text{ bits}$$

Nonuniform distribution has smaller entropy than uniform distribution

# Transmitting Information to Receiver

Random variable  $x$  that has eight possible states  $\{a, b, c, d, e, f, g, h\}$  with respective probabilities  $\{1/2, 1/4, 1/8, 1/16, 1/64, 1/64, 1/64, 1/64\}$

- Can use 3-bit number
- Or take advantage of nonuniform distribution
  - Shorter codes for more probable events at expense of longer codes for less probable events
  - Aim to shorten average code length
    - Example: Representing states  $\{a, b, c, d, e, f, g, h\}$  by code strings
      - 0, 10, 110, 1110, 111100, 111101, 111110, 111111
    - Average length of code to be transmitted:

Average code length

$$\frac{1}{2} \times 1 + \frac{1}{4} \times 2 + \frac{1}{8} \times 3 + \frac{1}{16} \times 4 + 4 \times \frac{1}{64} \times 6 = 2 \text{ bits}$$

Same as entropy of  $x$  in previous slide

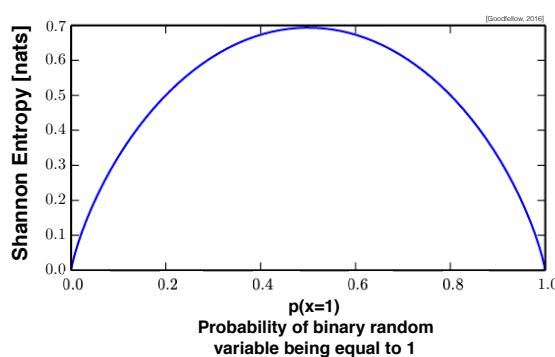
# Entropy and Coding Length

- General relation between entropy and shortest coding length
- Noiseless Coding Theorem (Shannon, 1948)
  - Entropy is a lower bound on number of bits needed to transmit state of a random variable

Units of entropy depend on type of logarithm used in defining it  
BITS if  $\log_2$       NATs if  $\log_e, \ln$

## Entropy of Binary Random Variable

- Random variable  $x$ , binary distribution  $p(x)$
- Entropy:  $(p - 1) \log(1 - p) - p \log p$
- Distribution is nearly deterministic when
  - $p$  near 0 (because  $x$  is nearly always 0)
  - $p$  near 1 (because  $x$  is nearly always 1)
- Maximal entropy when  $p = 0.5$  (because distribution of  $x$  is uniform)



# Entropy of Binary Random Variable

- Shannon entropy

$$H(x) = \mathbb{E}_{x \sim P}[I(x)] = -\mathbb{E}_{x \sim P}[\log P(x)]$$

- Differential entropy (continuous variable)

$$H[x] = - \int p(x) \ln p(x) dx$$

- Joint distribution  $p(x, y)$  from pairs of  $x$  and  $y$  values are drawn

- If value of  $x$  is known, average additional information needed to specify  $y$ :

Conditional  
entropy

$$H[y|x] = - \iint p(y, x) \ln p(y|x) dy dx$$

$H[x, y]$  — differential entropy of  $p(x, y)$

$H[x]$  — differential entropy of marginal  
 $p(x)$

$$H[x, y] = H[y|x] + H[x]$$

# Kullback-Leibler Divergence

- Relative entropy between distributions  $p(x)$  and  $q(x)$
- Consider unknown distribution  $p(x)$
- Model approximating distribution  $q(x)$
- If use  $q(x)$  to construct coding scheme to transmit values of  $x$
- Average additional amount of information needed to specify value of  $x$  if using  $q(x)$  instead of  $p(x)$ :

$$\begin{aligned} KL(p\|q) &= - \int p(x) \ln q(x) dx - \left( - \int p(x) \ln p(x) dx \right) \\ &= - \int p(x) \ln \left\{ \frac{q(x)}{p(x)} \right\} dx \end{aligned}$$

- In case of discrete variables:

- $KL(P||Q)$  is extra information needed to send message containing symbols drawn from probability distribution  $P$ , when we using code designed to minimize length of messages drawn from distribution  $Q$

# Kullback-Leibler Divergence Properties

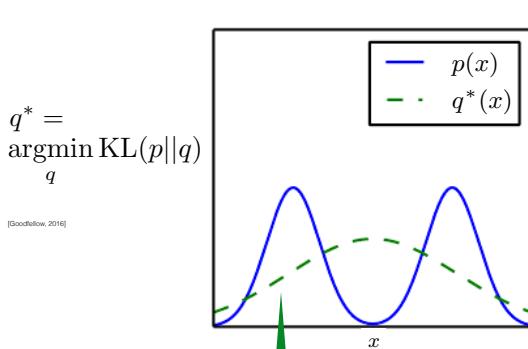
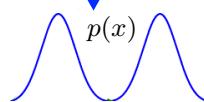
- Non-negative
  - Discrete variables:
    - ♦  $\text{KL}(P||Q) = 0$  iff P and Q are the same distribution in case of discrete variables
  - Continuous variables
    - ♦  $\text{KL}(P||Q) = 0$  iff P and Q are equal *almost everywhere*
- Measures difference between two distributions
- KL often thought of as measure of distance between distributions
  - But is not really distance — because it is *not* symmetric

$$\text{KL}(P||Q) \neq \text{KL}(Q||P)$$

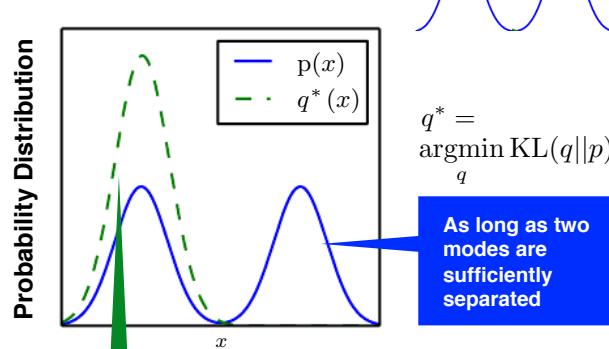
## Kullback-Leibler Divergence Asymmetry

- Approximating  $p(x)$  with another distribution  $q(x)$ 
  - Minimizing  $\text{KL}(p||q)$  or  $\text{KL}(q||p)$  depends on application needs — results in approximation that tends to
    - ♦ Place high probability wherever true distribution has high probability
    - ♦ Avoid high probability where true distribution has low probability

- Example:
- Mixture of two Gaussians
  - Bimodal



To put significant probability mass on all modes, q blurs modes together



To avoid probability mass in low-probability areas between modes of p, distribution q chooses one of modes

# Cross-Entropy

- **Narrow definition of cross-entropy:**
  - Often defined as negative log-likelihood of Bernoulli (or softmax)
- **More broadly:**
  - Cross-entropy entropy between empirical distribution (defined by training set) and model
    - Any loss consisting of negative log-likelihood
    - Example:
      - MSE is cross-entropy between empirical distribution and Gaussian model
- **Therefore, maximum likelihood:**
  - Make model distribution match empirical distribution  $\hat{p}_{data}$ 
    - Since cannot have *true* data-generating distribution  $p_{data}$

J. Braun

15

## Cross-Entropy and KL Divergence

- Now consider cross-entropy from information-theory perspective

$$H(P, Q) = -\mathbb{E}_{x \sim P} \log Q(x) - \int p(x) \ln q(x) dx$$

- Cross-entropy can be rewritten in terms of KL divergence

$$H(P, Q) = H(P) + \text{KL}(P||Q)$$

- Minimizing cross-entropy w.r.t.  $Q$  is equivalent to minimizing KL divergence (because  $P$  does not contribute to gradient)

J. Braun

16

# Maximum Likelihood and KL Divergence

Recall Max Likelihood

$$\theta_{\text{ML}} = \arg \max_{\theta} \sum_{i=1}^m \log p_{\text{model}}(\mathbf{x}^{(i)}; \theta)$$

Rescaling does not affect maximization  
So, rescale by 1/m

$$\theta_{\text{ML}} = \arg \max_{\theta} \mathbb{E}_{\mathbf{x} \sim \hat{p}_{\text{data}}} \log p_{\text{model}}(\mathbf{x}; \theta)$$

Now consider minimizing KL divergence w.r.t.  $p_{\text{model}}$

$$\text{KL}(\hat{p}_{\text{data}} \| p_{\text{model}}) = \mathbb{E}_{\mathbf{x} \sim \hat{p}_{\text{data}}} [\log \hat{p}_{\text{data}}(\mathbf{x}) - \log p_{\text{model}}(\mathbf{x})]$$

Function of only data generation, not of model

And argmin of this term (note minus sign) is equivalent to argmax of  $\theta_{\text{ML}}$

Training to minimize KL divergence maximizes  $\theta_{\text{ML}}$

# Mutual Information

- Two random variables (x,y) NOT independent
- Quantify “degree” of independence by KL divergence between joint distribution and product of the marginals, given by

Mutual Information

$$\begin{aligned} I[\mathbf{x}, \mathbf{y}] &\equiv \text{KL}(p(\mathbf{x}, \mathbf{y}) \| p(\mathbf{x})p(\mathbf{y})) \\ &= - \iint p(\mathbf{x}, \mathbf{y}) \ln \left( \frac{p(\mathbf{x})p(\mathbf{y})}{p(\mathbf{x}, \mathbf{y})} \right) d\mathbf{x} d\mathbf{y} \end{aligned}$$

$$I[\mathbf{x}, \mathbf{y}] \geq 0$$

Iff x and y independent

$$I[\mathbf{x}, \mathbf{y}] = 0$$

$$I[\mathbf{x}, \mathbf{y}] = H[\mathbf{x}] - H[\mathbf{x}|\mathbf{y}] = H[\mathbf{y}] - H[\mathbf{y}|\mathbf{x}]$$

---

# **Questions?**

---

# **Neural Networks and Deep Learning**

**Dr. Jerome J. Braun**

## **This Lecture: Supervised-Learning Basics V**

Course: Neural Networks and Deep Learning  
IE 7615

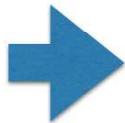
---

---

**Unauthorized distributing, redistributing, posting,  
and/or reposting, of the materials of this course  
(including course lectures, lecture slides, slide-sets,  
syllabi, guidelines, assignments, quizzes, exams,  
presentations, software, electronic media, etc.),  
is prohibited.**

# This Lecture

---



- Generative vs. Discriminative Approaches
- Generalized Linear Models
- Linear Discriminant Analysis (LDA)

## Three Approaches to Classification (1)

---

- **Generative**
  - First solve inference problem of determining **class-conditional densities**  $p(x|C_k)$  for each class  $C_k$  individually
  - Separately infer prior **class probabilities (class priors)**  $p(C_k)$
  - Then use Bayes' theorem to find **class posteriors**  $p(C_k|x)$

$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\mathbf{x})} \quad p(\mathbf{x}) = \sum_k p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)$$

## Three Approaches to Classification (2)

- **Discriminative**

- First solve inference problem of determining posterior class probabilities  $p(C_k|x)$

- Then use decision theory to assign each new  $x$  to one of  $K$  classes

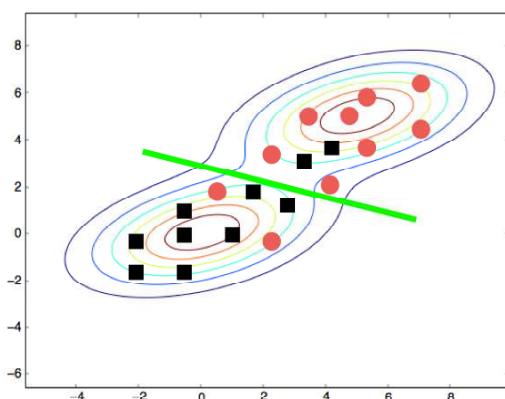
- Find **discriminant function  $f(x)$**

- $f(x)$  maps each input  $x$  directly onto class label

- **Probabilities play no role**

J. Braun

## Generative vs. Discriminative



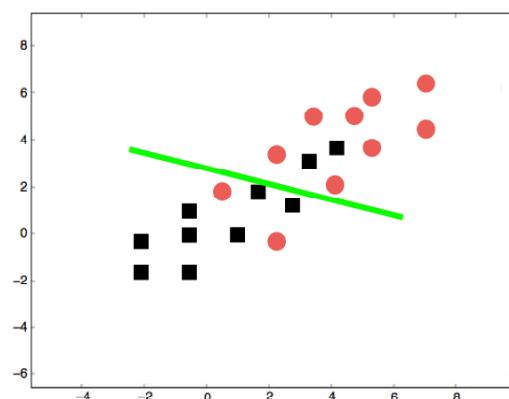
Determine class-conditional distributions

Derive decision boundary based on above

YES

Ability to sample from distributions?

No



Determine decision boundary directly

No distribution determination burden

J. Braun

# This Lecture

- Generative vs. Discriminative Approaches
- Generalized Linear Models
- Linear Discriminant Analysis (LDA)

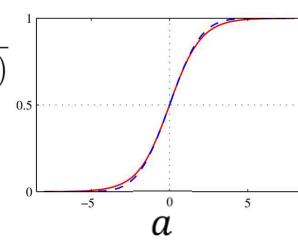
J. Braun

7

## Modeling $P(C_k|x)$ as Sigmoid

Sigmoid function (a.k.a. logistic or “squashing” function)

$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$



$$\begin{aligned} p(\mathcal{C}_1|\mathbf{x}) &= \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1) + p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2)} \\ &= \frac{1}{1 + \exp(-a)} = \sigma(a) \end{aligned}$$

where

“Log-odds”  $\rightarrow a = \ln \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2)}$

When  $a \geq 0$ ,  $\sigma(a) \geq 0.5$

$a \geq 0$  equivalent to  $p(C_1|x) \geq p(C_2|x)$

Decision

8

# Generalized Linear Models

---


$$a = \ln \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2)}$$

- If  $a(x)$  is linear function of  $x$ , then have **generalized linear model**

- **Generative**

- Gaussian distribution with same covariance for class-conditional densities
  - Leads to Linear Discriminant Analysis — LDA (next)

- **Discriminative**

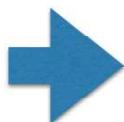
- Fit linear function  $a(x)$  
$$a(x) = \theta^T x$$
  - E.g., Logistic Regression

---

9

## This Lecture

---



- **Generative vs. Discriminative Approaches**
- **Generalized Linear Models**
- **Linear Discriminant Analysis (LDA)**

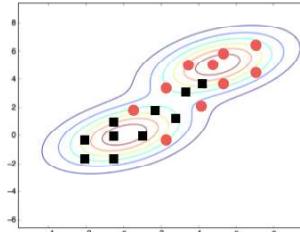
# Two-class Problem

- Consider two-class problem

$$p(C_k) = \pi_k$$

$$\pi_k \geq 0$$

$$\sum_{k=1}^2 \pi_k = 1$$



- Recall Gaussian distribution

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\boldsymbol{\Sigma}|^{1/2}} \exp \left\{ -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right\}$$

Dimensionality

Covariance matrix

( $\mathbf{x}$  and  $\boldsymbol{\mu}$  column vectors)

11

# Using Gaussian Density and LDA

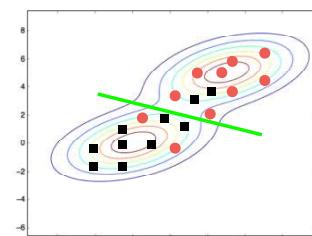
- Assume  $p(\mathbf{x}|C_k)$  Gaussian for each class

$$f_k(\mathbf{x}) = \frac{1}{(2\pi)^{p/2} |\boldsymbol{\Sigma}_k|^{1/2}} e^{-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k)}$$

Dimensionality

Covariance matrix

( $\mathbf{x}$  and  $\boldsymbol{\mu}$  column vectors)



$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\boldsymbol{\Sigma}|^{1/2}} \exp \left\{ -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right\}$$

Now assume covariances are same for both classes

$\forall k : \boldsymbol{\Sigma}_k = \boldsymbol{\Sigma}$

Under these assumptions problem becomes linear

## Decision Function

- Pick class which has maximum posterior probability, given feature vector  $x$

$$\operatorname{argmax}_k P(C = C_k) | X = x =$$

$$= \operatorname{argmax}_k f_k(x)\pi_k =$$

$$= \operatorname{argmax}_k \log(f_k(x)\pi_k) =$$

$$= \operatorname{argmax}_k \left[ -\log((2\pi)^{\frac{p}{2}} |\Sigma|^{\frac{p}{2}}) - \frac{1}{2}(x - \mu_k)^T \Sigma^{-1} (x - \mu_k) + \log(\pi_k) \right] =$$

$$= \operatorname{argmax}_k \left[ -\frac{1}{2}(x - \mu_k)^T \Sigma^{-1} (x - \mu_k) + \log(\pi_k) \right] =$$

13

## Linear Decision Function

- Expanding terms of decision function  $a(x)$

$$-\frac{1}{2}(x - \mu_k)^T \Sigma^{-1} (x - \mu_k) =$$

$$x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k - \frac{1}{2} x^T \Sigma^{-1} x$$

- Leads to simpler decision function

Ex.

- Linear in  $x$ :

$$\operatorname{argmax}_k \left[ x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log(\pi_k) \right]$$

14

# Linear Discriminant

Linear Discriminant Function

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log(\pi_k)$$

$$\operatorname{argmax}_k \left[ x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log(\pi_k) \right] = \operatorname{argmax}_k \delta_k(x)$$

Decision boundary between classes  $k$  and  $l$

$$\{x : \delta_k(x) = \delta_l(x)\}$$

- Equivalently log-odds function must be 0

$$\log \frac{\pi_k}{\pi_l} - \frac{1}{2} (\mu_k + \mu_l)^T \Sigma^{-1} (\mu_k - \mu_l) + x^T \Sigma^{-1} (\mu_k - \mu_l) = 0$$

15

# Decision Boundary

$$\log \frac{\pi_k}{\pi_l} - \frac{1}{2} (\mu_k + \mu_l)^T \Sigma^{-1} (\mu_k - \mu_l) + x^T \Sigma^{-1} (\mu_k - \mu_l) = 0$$

Class prior log-ratio

Constant

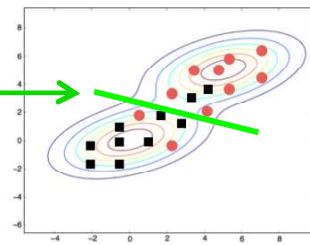
Input

Covariance

Difference of class means

Can be rewritten as

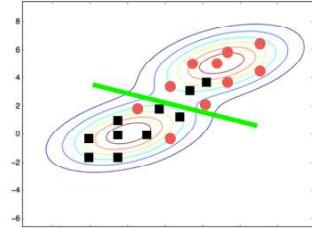
$$\theta_0 + x^T \theta = 0$$



16

# Linear Discriminant Analysis

- **Assume**  $p(C_k) = \pi_k$      $\sum_{k=1}^2 \pi_k = 1$   
 $\pi_k \geq 0$
- **$p(x|C_k)$  assumed Gaussian**
- $f_k(x) = \frac{1}{(2\pi)^{p/2} |\Sigma_k|^{1/2}} e^{-\frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1} (x-\mu_k)}$



Equal covariances  
assumed

$$\forall k : \Sigma_k = \Sigma$$

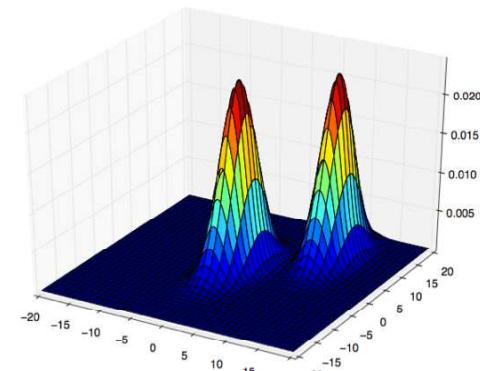
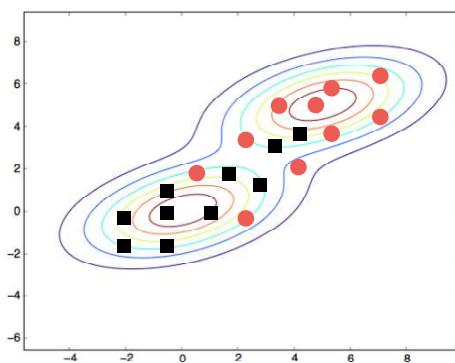
Under these assumptions problem linear

$$\log \frac{\pi_k}{\pi_l} - \frac{1}{2}(\mu_k + \mu_l)^T \Sigma^{-1} (\mu_k - \mu_l) + x^T \Sigma^{-1} (\mu_k - \mu_l) = 0$$

17

## Effect of Covariance Matrix

- Covariance matrix determines shape of Gaussian distribution
- In LDA
  - Covariances equal — same shapes
  - Means different — distributions shifted



18

# Effect of Class Priors

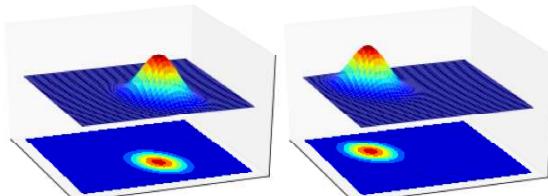
- Differences in class priors
  - Affect decision boundary/hypersurface location w.r.t. class means location
  - Via class prior log-ratio  $\log \frac{\pi_k}{\pi_l}$
- If priors not equal
  - Decision boundary located further away from mean of higher-prior class

$$\log \frac{\pi_k}{\pi_l} - \frac{1}{2}(\mu_k + \mu_l)^T \Sigma^{-1}(\mu_k - \mu_l) + x^T \Sigma^{-1}(\mu_k - \mu_l) = 0$$

## Effect of Class Priors (cont.)

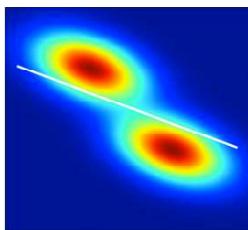
Example: two classes  $C_1$  and  $C_2$

- Gaussian  $p(C_1|x)$  and  $p(C_2|x)$ 
  - Same covariance:  $S$
  - Different means:  $m_1, m_2$

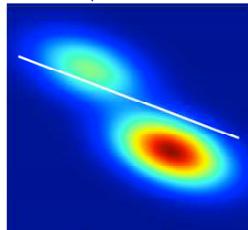


$$\log \frac{\pi_k}{\pi_l} - \frac{1}{2}(\mu_k + \mu_l)^T \Sigma^{-1}(\mu_k - \mu_l) + x^T \Sigma^{-1}(\mu_k - \mu_l) = 0$$

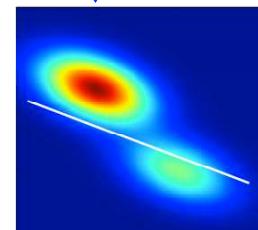
Equal priors



Unequal priors,  $\pi_1 < \pi_2$



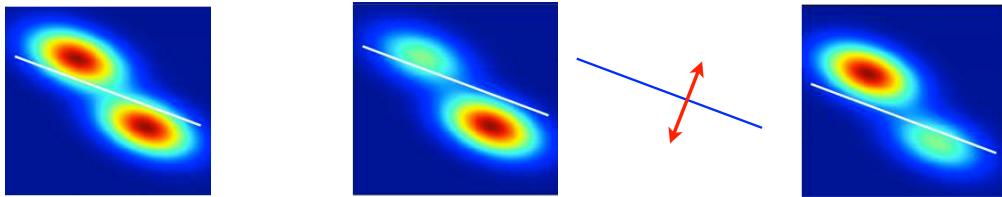
Unequal priors,  $\pi_1 > \pi_2$



Decision boundary located further away from mean of higher-prior class

## Effect of Class Priors (cont.)

- Differences in class priors
  - Affect decision boundary/hypersurface location w.r.t. class means location
  - Via class prior log-ratio  $\log \frac{\pi_k}{\pi_l}$
- If priors not equal
  - Decision boundary located further away from mean of higher-prior class



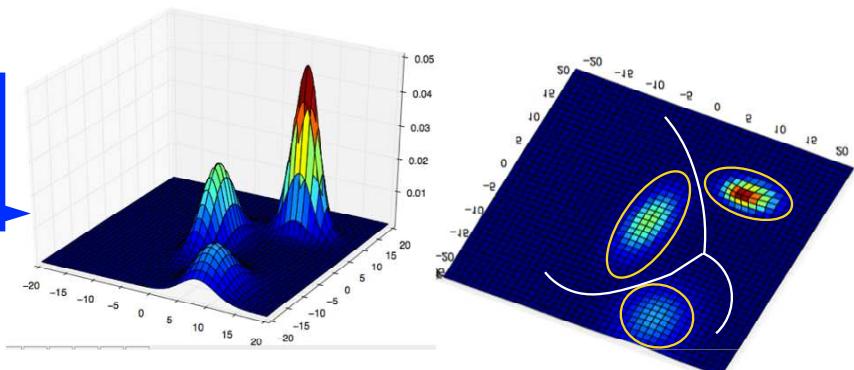
J. Braun

21

## Unequal Covariances

- When class covariance-matrices are not equal
  - Decision boundaries (decision hypersurfaces) are no longer linear

Example:  
• Three classes  
• Covariance matrices of classes not equal



J. Braun

22

# **Questions?**

---

# **Neural Networks and Deep Learning**

**Dr. Jerome J. Braun**

## **This Lecture: Regularization**

Course: Neural Networks and Deep Learning  
IE 7615

---

---

**Unauthorized distributing, redistributing, posting,  
and/or reposting, of the materials of this course  
(including course lectures, lecture slides, slide-sets,  
syllabi, guidelines, assignments, quizzes, exams,  
presentations, software, electronic media, etc.),  
is prohibited.**

# This Lecture

---



**Regularization**

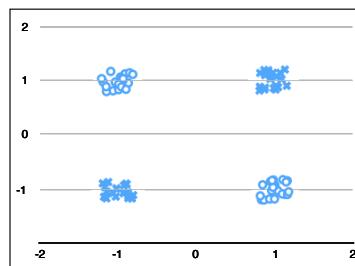
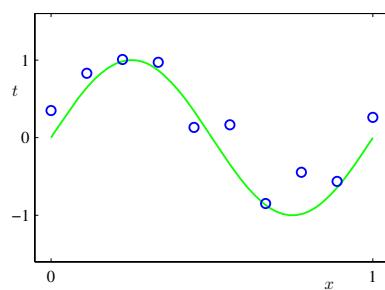
**Overfitting**

**Brief notion of capacity control**

# Classification of Nonlinear Data

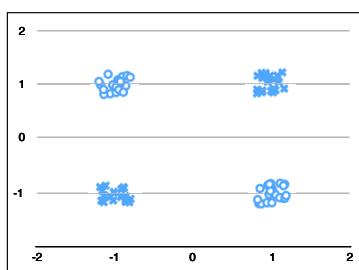
---

- **Examples**

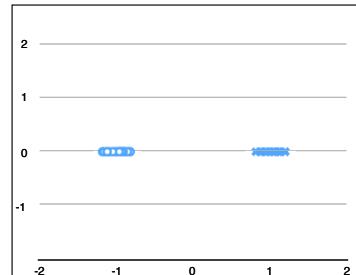


# Nonlinear Basis Functions

- Transforming inputs (features)
  - May result in linear separability
- Example



$$\varphi(x) : x \in R^2 \rightarrow z = x_1 \cdot x_2$$

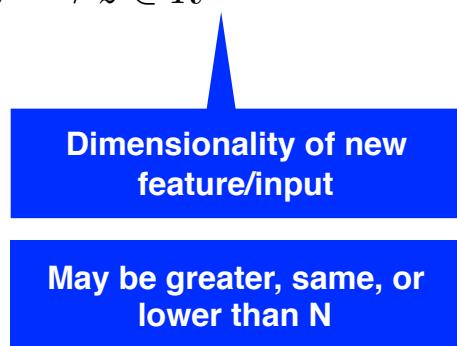


5

# Nonlinear Mapping

- Nonlinear mapping in general

$$\varphi(x) : x \in R^N \rightarrow z \in R^M$$

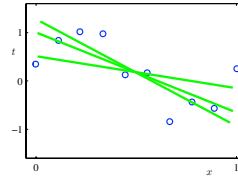


6

# Choosing Model

- **$N=10$  points**

- **Data**



- 

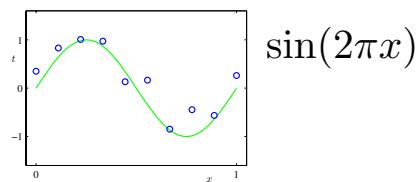
J. Braun

7

# Nonlinear Data

- **$N=10$  points**

- **True model — sinusoidal**



[Bishop]

- **Polynomial basis functions**

- 

$$\phi = [1, x, x^2 \dots, x^M]^T$$

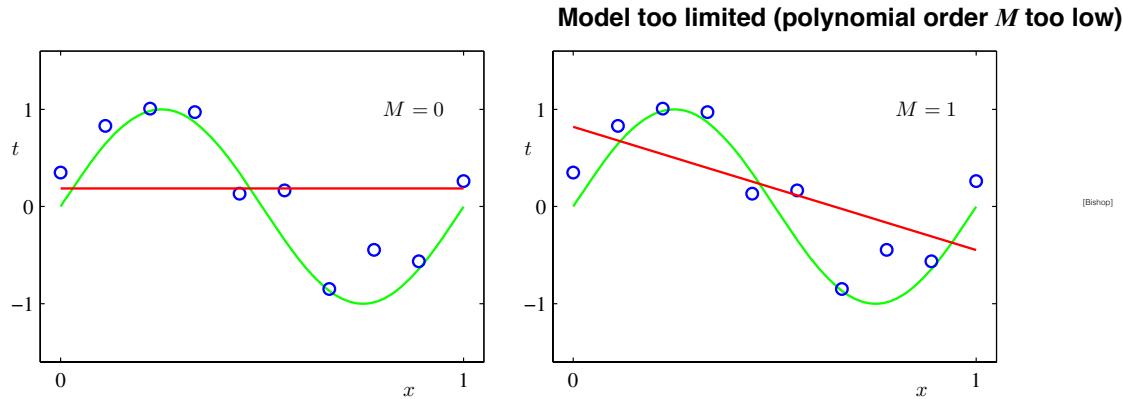
J. Braun

8

## Representational Capacity

### Linear Regression Example

- Too low
- Under-fitting



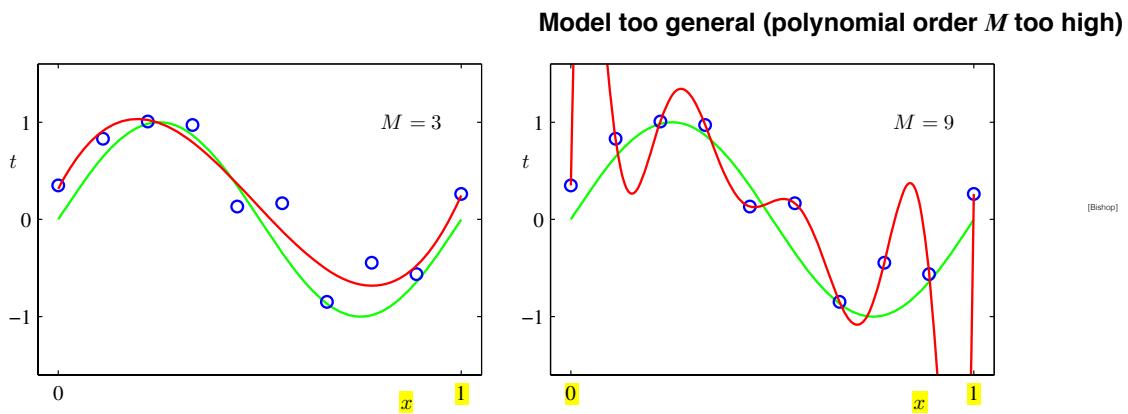
J. Braun

9

## Representational Capacity

### Linear Regression Example

- Too high
- Overfitting



- Better results on training set, poor generalization

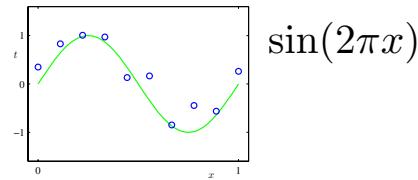
J. Braun

10

# Nonlinear Data

- $N=10$  points

- True model — sinusoidal



- Polynomial basis functions

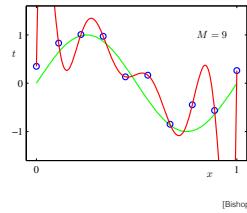
$$\phi = [1, x, x^2 \dots, x^M]^T$$

J. Braun

11

# Overfitting

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
$w_0^*$	0.19	0.82	0.31	0.35
$w_1^*$		-1.27	7.99	232.37
$w_2^*$			-25.43	-5321.83
$w_3^*$			17.37	48568.31
$w_4^*$				-231639.30
$w_5^*$				640042.26
$w_6^*$				-1061800.52
$w_7^*$				1042400.18
$w_8^*$				-557682.99
$w_9^*$				125201.43



- For larger  $M$

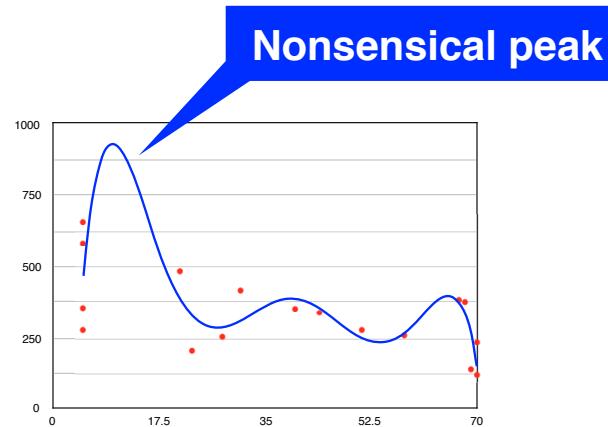
- Large weight coefficient values

J. Braun

12

# Overfitting Example

- Gembrozil clearance example from previous lectures
  - M=6



# Overfitting Mitigation

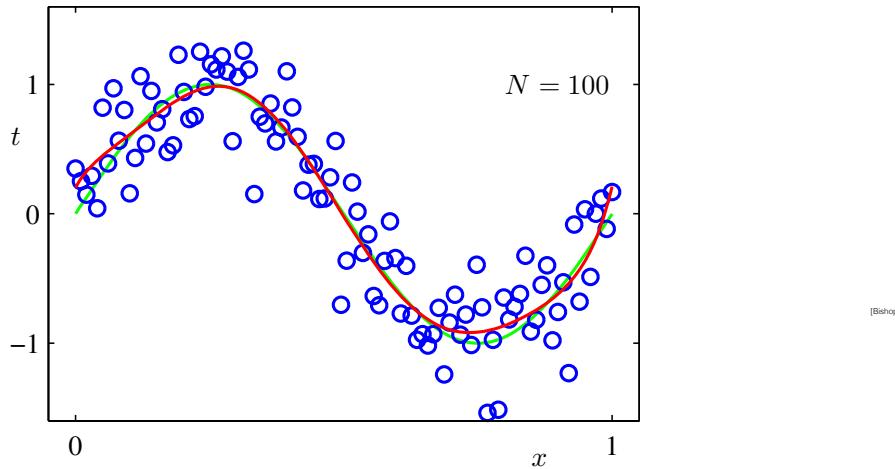
- Potential directions
  - More training data
  - Cross-validation
  - Regularization, e.g., weight decay
  - Regularizing effects in neural networks, with, e.g.,
    - Dropout
    - Specialized regularizers in recurrent networks
    - Batch normalization
    - and other

We will  
see  
these  
later in  
this  
course

## Training Set Size

### Linear Regression Example

- Increasing number of exemplars helps prevent under- or over-fitting

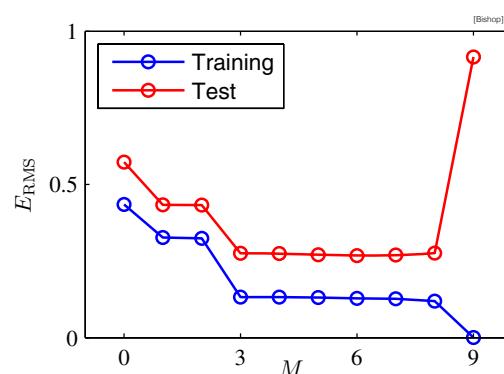


- But more data may be unavailable

## Model Complexity and Performance

### Linear Regression Example

- Root mean square error
  - RMS error (RMSE)
- As model complexity increases
  - Up to *some* threshold, performance improves on both training and test data
  - Beyond *some* threshold, performance
    - On training data increases
    - On test data decreases!



# No Free Lunch Theorem

- Learning theory objective (claim):
  - Generalize well from finite set of examples
  - Seems to contradict certain principles of logic
    - Inductive reasoning: inferring general rules from limited set of examples not logically valid
      - to infer rule describing every element of set, must have some info about each element
- Machine learning avoid above — only *probably* correct rules
  - but:
- NFLT [Wolpert, 1996]:
  - Averaged over *all possible* data distributions, same error rate on previously unseen data
  - No universally “best” or “better” algorithm
  - Task dependence
  - Motivation for regularization

# Norms

- Functions of vectors or matrices

- Represent magnitude

$$\|x\|_p = \left( \sum_i |x_i|^p \right)^{\frac{1}{p}}$$

- **$L^p$  norms of vectors**

- $L^2$  most usual

$$\|x\|_2 = \sqrt{\sum_i |x_i|^2} \quad (\text{abbreviated as } \| \| )$$
$$\|x\|_\infty = \max_i |x_i|$$

- Norms of matrices

- Frobenius norm

$$\|A\|_F = \sqrt{\sum_{i,j} A_{ij}^2}$$

# Regularization

---

- **Empirical risk**  $R_{\text{emp}}$ 
  - $R$ : **risk functional**
- **$\lambda$ : regularization parameter**

$$\lambda > 0$$

- $\Omega$ : **Regularization term (regularizer)**
- **Tradeoff between minimizing  $R_{\text{emp}}$  and model simplicity (small  $\Omega$ )**

## Weight Decay Regularizer

---

- **Weight decay regularizer**

$$\Omega[f] = \mathbf{w}^T \mathbf{w} = \|\mathbf{w}\|_2^2 = \|\mathbf{w}\|^2$$

- **In linear regression**

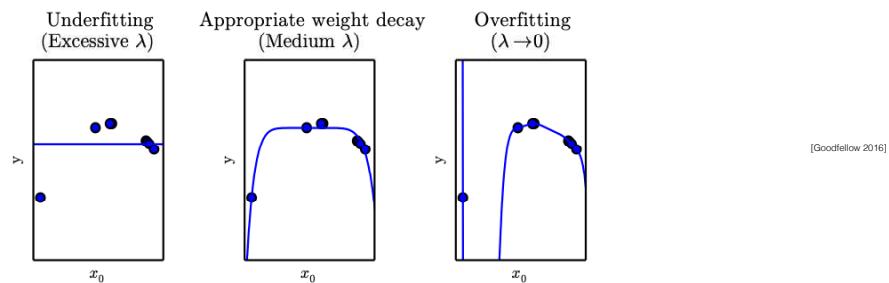
$$J(\mathbf{w}) = \text{RMSE}_{\text{train}} + \lambda \mathbf{w}^T \mathbf{w}$$

Penalizes large weight values

# Regularized Linear Regression

- **Choosing  $\lambda$**

- **Control of regularization level**
- **“Flattest” function that represents training data sufficiently well**



J. Braun

21

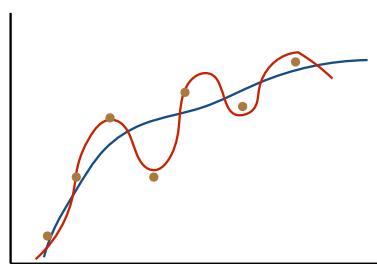
# Regularized Linear Regression

$$J(\theta) = \frac{1}{2m} \left[ \sum_{i=1}^m \left( h_{\theta}(x^{(i)}) - y^{(i)} \right)^2 + \lambda \sum_{j=1}^n \theta_j^2 \right]$$

Regularization parameter and regularizer

↓

$$\operatorname{argmin}_{\theta} J(\theta)$$



22

# Regularized Gradient Descent

Repeat {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

For Linear  
Regression

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \quad (j = 1, 2, 3, \dots, n)$$

}



$$\theta_j := \theta_j \left(1 - \alpha \frac{\lambda}{m}\right) - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Exercise

- Similar form for Logistic Regression
- Hypothesis for Logistic Regression different

# Regularized Normal Equation

$$\theta = (X^T X)^{-1} X^T y$$

$$\theta = \left( X^T X + \lambda \begin{bmatrix} 0 & \dots & \dots \\ \dots & 1 & \dots \\ \dots & \dots & 1 \dots \\ & \dots & \dots & 1 \end{bmatrix} \right)^{-1} X^T y$$

# Model Complexity and Determination of $\lambda$

---

- **Regularization allows to train complex models using limited-size datasets**
  - Without severe overfitting
  - Limits effective model complexity
- **Regularization shifts determination of best model-complexity**
  - From finding appropriate number of basis functions
  - To determining suitable value of regularization coefficient  $\lambda$

We will discuss regularization for deep networks later in this course

Questions?

---

# **Neural Networks and Deep Learning**

**Dr. Jerome J. Braun**

## **This Lecture: Artificial Neural Networks — Fundamentals I**

Course: Neural Networks and Deep Learning  
IE 7615

---

---

**Unauthorized distributing, redistributing, posting,  
and/or reposting, of the materials of this course  
(including course lectures, lecture slides, slide-sets,  
syllabi, guidelines, assignments, quizzes, exams,  
presentations, software, electronic media, etc.),  
is prohibited.**

# This Lecture

---

- Artificial Neural Networks (ANN) history and overview
- ANN basics

## ANN Application Areas (Recap)

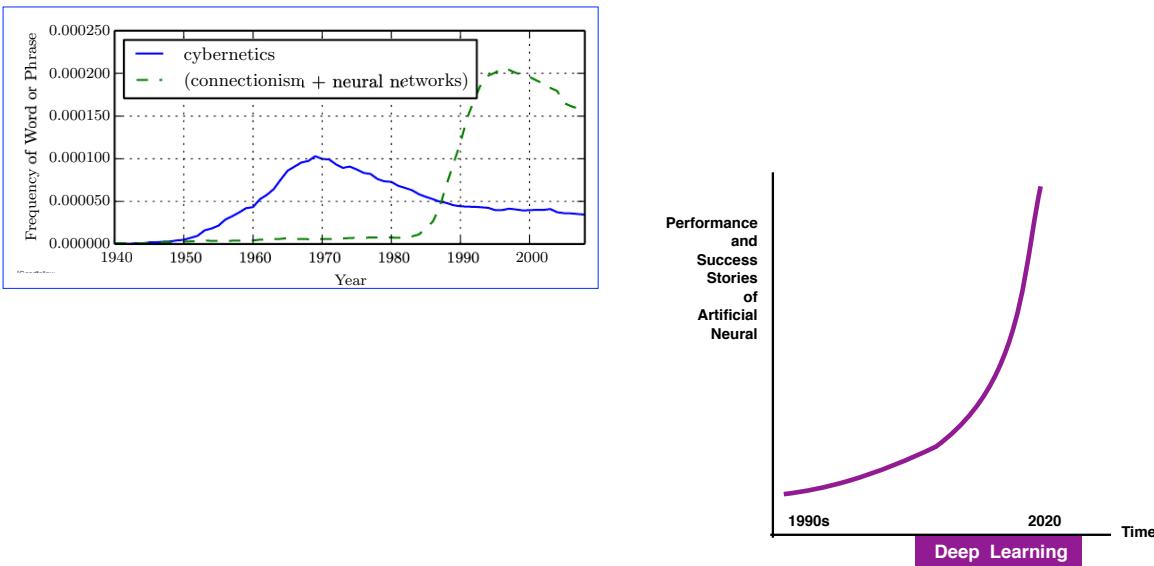
---

Recall

A few examples (out of many)	
Machine Vision	Speech & Language Processing (NLP)
Face Recognition, Object Recognition	Autonomous Vehicles
Self-driving cars	Robotics
Forecasting	Assistive Robotics
Weather, climate, etc.	Bioinformatics
Computational biology, neuroscience	"X-informatics
Analytics (text, video, etc.)	Economics, Financial forecasting
Medical diagnostics	Insurance
Biomed data analysis, healthcare	Fraud detection
Drug development	Image/photo tagging
Personalized medicine	
... many others...	

- For many problems and application, at this time ANNs appear to be best (or most promising) solution-path
- For some problems and applications, at this time ANNs are the **only** viable solution-path

# ANN “Waves of Evolution”



J. Braun

5

# History of Artificial Neural Networks

- “Early Days” →
  - 1943: McCulloch and Pitts — McCulloch-Pitts neuron model
  - 1949: Hebb — Hebbian learning rule
  - 1958: Rosenblatt — Perceptron
  - 1969: Minsky and Papert — perceptron limitations (*Perceptrons*)
  - 1980: Grossberg — Adaptive Resonance Theory (ART)
  - 1982: Hopfield — Hopfield networks
  - 1982: Kohonen — Self-Organizing Maps
- “Ban” →
  - 1986 Back-propagation for Multi-Layer Perceptrons (resurfaced)
  - Late 1980’s — Now: Aggressive progress of ANN field
  - Late 1990s — “Competition” (SVMs, also HMMs, BNs, etc.)
- “Renaissance” →
  - 2000+ — Now: Deep Networks (and recurrent networks)
    - + Hinton, others
- “Boom” →

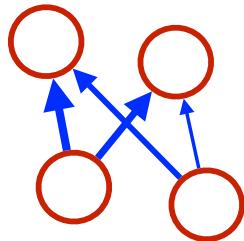
J. Braun

6

## What is Artificial Neural Network (ANN)? — Recap

---

- **Units** with associated function
  - Also referred to in literature as nodes or “neurons”
- **Connections** — weighted **links** between units
- **Learning algorithm**
  - Typically — modify weights of connections
  - Sometimes — modify units, network topology
- **Conventional artificial neural networks (ANNs) are NOT models of biological neural networks or brain !**
  - We will see this later in this course



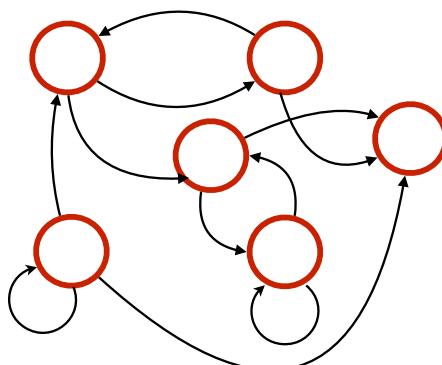
J. Braun

7

## ANN Topologies (Recap)

---

- **Most general — fully/arbitrarily connected**



J. Braun

8

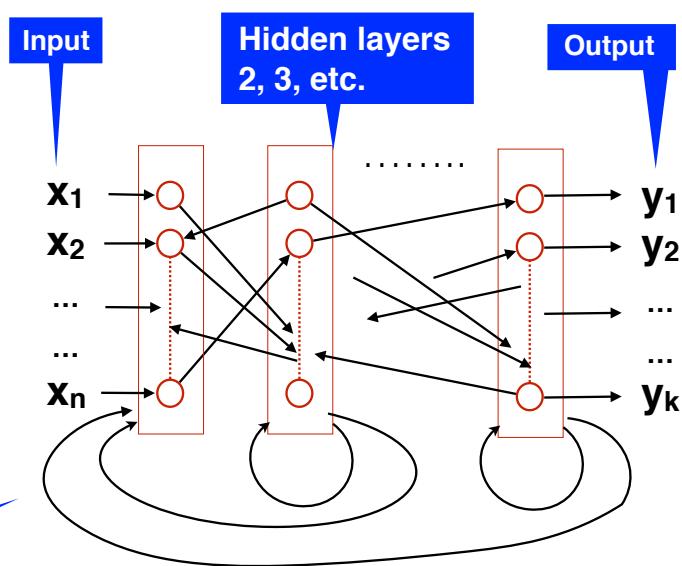
## List of Some ANN Architecture Categories (Recap)

- “Shallow” multilayer perceptron (MLP) networks
- Deep MLP networks
- Convolutional networks
- Recurrent networks, LSTMs, GRUs, ...
- Autoencoders
- Transformers, BERT, ...
- Generative adversarial networks (GANs)
- ... and other

We will study them  
in this course

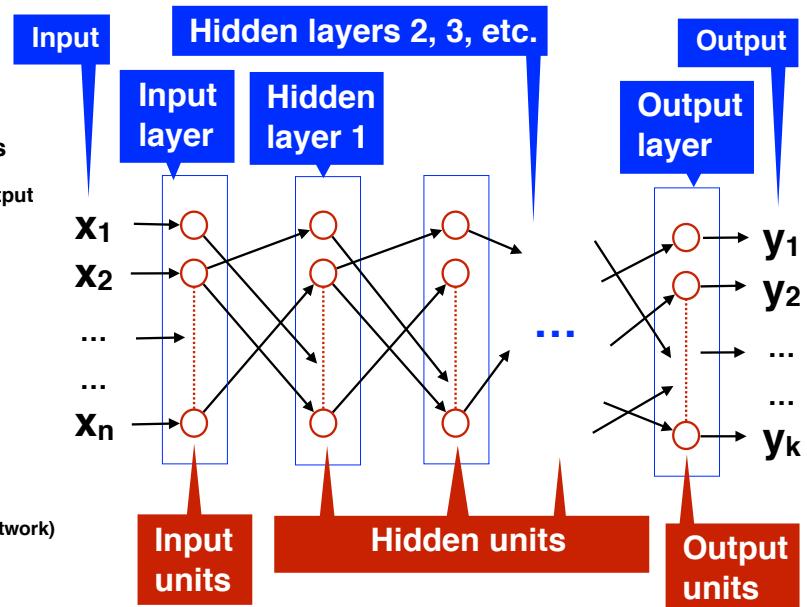
## ANN Topologies – Layered (Recap)

- Layered
- Any number of layers
- Retrograde connections
  - Within layers
  - To previous layers



## ANN Topologies – Feedforward (Recap)

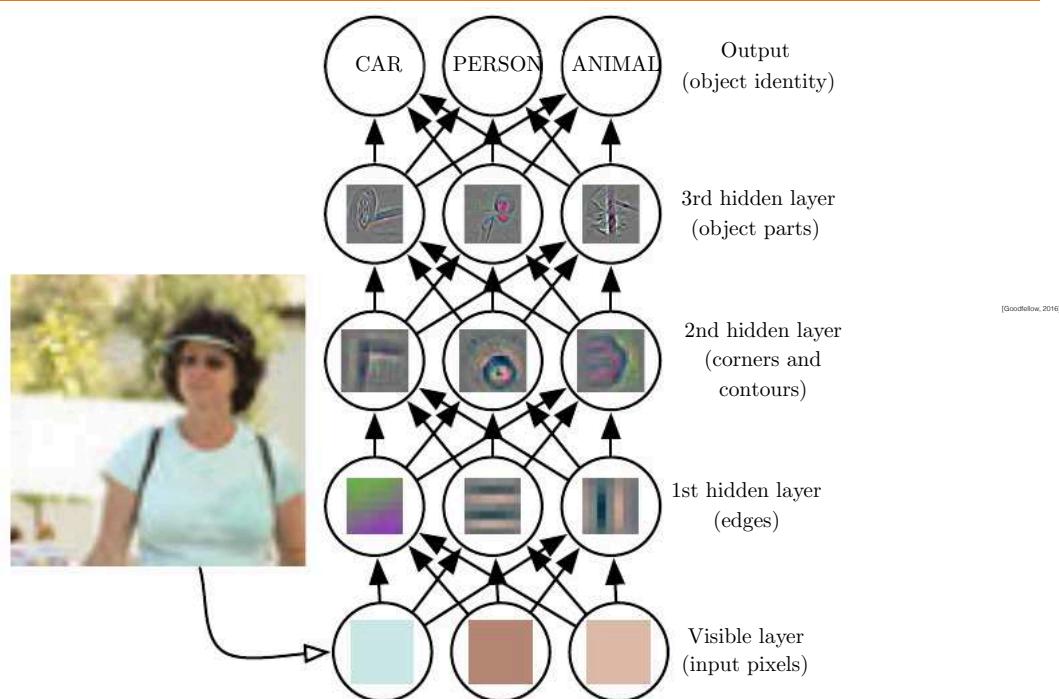
- Layered
- Only forward connections
  - Direction from input to output
  - From layer  $i$  to layer  $i + 1$
- Any number of layers
- Layers
  - Input layer
  - Hidden layers
    - One (shallow network)
    - More than one (deep network)
  - Output layer



J. Braun

11

## Deep Learning Model (Recap)

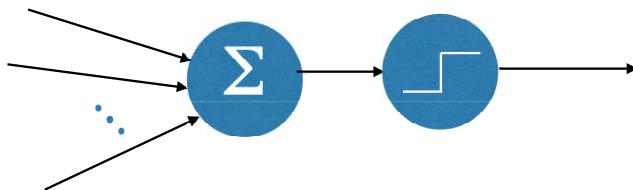


12

## Recap: Early “Neuron Models” for ANNs

---

- McCulloch-Pitts neuron model
  - Simple
  - Computationally efficient
  - No neurobiological plausibility



## Early Developments: Hebbian Learning

---

- D. Hebb (1949)
  - Premise: If neurons on both sides of synapse are activated synchronously and repeatedly, synaptic strength is selectively increased

# Recap: Perceptron and Adaline

- **Rosenblatt (1958):**

- “machine that learns, using examples, to assign input vectors (samples) to different classes using a linear function of the inputs”

$$y(\mathbf{x}) = f(\mathbf{w}^T \phi(\mathbf{x}))$$

- **Minsky and Papert (1969):**

- “stochastic gradient-descent algorithm that attempts to linearly separate a set of n-dimensional training data”

$$f(a) = \begin{cases} +1, & a \geq 0 \\ -1, & a < 0 \end{cases}$$

- **ADALINE (Widrow-Hoff)**

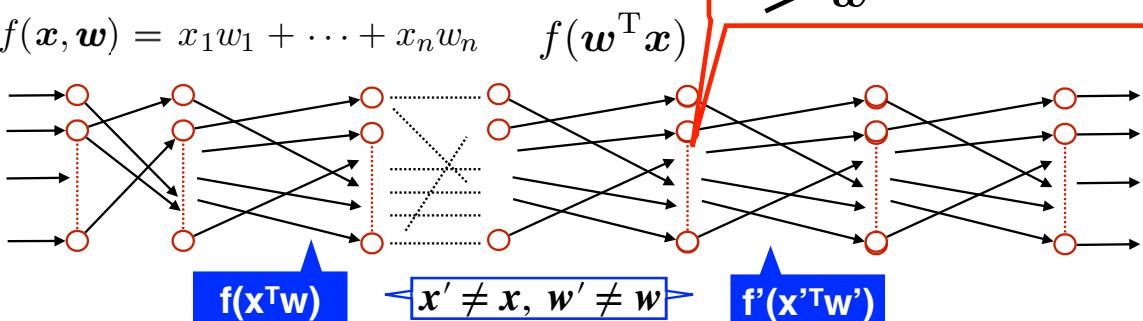
- ADAptive LInear Element
- Single processing unit
- Threshold non-linearity

## Linear Constructs in ANNs (Recap)

Linear constructs (such as those we see in linear regression) appear in neural networks (ANNs)

$$f(x, \mathbf{w}) = x_1 w_1 + \dots + x_n w_n$$

$$f(\mathbf{w}^T \mathbf{x})$$



Note *linear constructs* as “elements” inside ANN

Hence, need to understand (study) linear models

Although, as we will see, ANNs are NOT linear models!

# Linear Discriminant Function Geometry

$$y(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$$

- Consider two points  $\mathbf{x}_A, \mathbf{x}_B$
- If both  $\mathbf{x}_A, \mathbf{x}_B$  are on decision surface, must have  $y(\mathbf{x}_A) = 0$  and  $y(\mathbf{x}_B) = 0$

**Therefore**  $\mathbf{w}^T \mathbf{x}_A + w_0 = \mathbf{w}^T \mathbf{x}_B + w_0 = 0$

i.e.,  $\mathbf{w}^T (\mathbf{x}_A - \mathbf{x}_B) = 0$

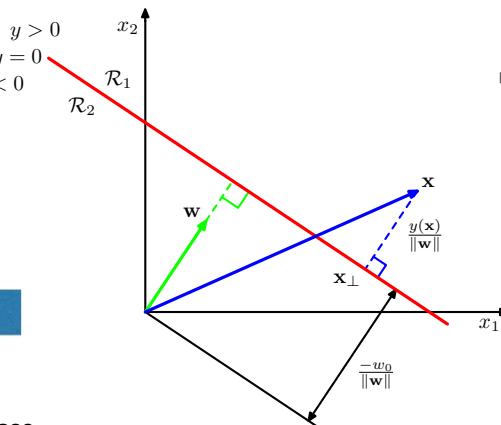
- Hence  $\mathbf{w}$  is orthogonal to every vector on decision surface
  - i.e.,  $\mathbf{w}$  determines orientation of decision surface
- Also, since for any  $\mathbf{x}$  on decision surface  $y(\mathbf{x})=0$ , normal distance from origin to decision surface is given by

$$\frac{\mathbf{w}^T \mathbf{x}}{\|\mathbf{w}\|} = -\frac{w_0}{\|\mathbf{w}\|}$$

- i.e., bias  $w_0$  determines location of decision surface

J. Braun

[Bishop]



17

## Distance of Points To Decision Surface (Linear Discriminant Function)

- Signed distance-measure of  $\mathbf{x}$  from decision surface is  $y(\mathbf{x})$ 
  - Why:

Denote orthogonal projection of  $\mathbf{x}$  onto decision surface as

so that

$$\mathbf{x} = \mathbf{x}_\perp + r \frac{\mathbf{w}}{\|\mathbf{w}\|}$$

Multiply above by  $\mathbf{w}^T$ , add  $w_0$  and use:

$$y(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0 \quad \text{and} \quad y(\mathbf{x}_\perp) = \mathbf{w}^T \mathbf{x}_\perp + w_0 = 0$$

yields  $\rightarrow r = \frac{y(\mathbf{x})}{\|\mathbf{w}\|}$

J. Braun

18

---

**... to be continued...**

---

# **Neural Networks and Deep Learning**

**Dr. Jerome J. Braun**

## **This Lecture: Tensors in Deep Learning, Computational Graphs I**

Course: Neural Networks and Deep Learning  
IE 7615

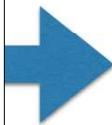
---

---

**Unauthorized distributing, redistributing, posting,  
and/or reposting, of the materials of this course  
(including course lectures, lecture slides, slide-sets,  
syllabi, guidelines, assignments, quizzes, exams,  
presentations, software, electronic media, etc.),  
is prohibited.**

# This Lecture

---



- Tensors in Deep Learning
- Computational Graphs I

## Tensors and Tensor Calculus

---

- **Tensors in physics, mathematics, etc.**
  - **Tensor calculus**
    - ♦ Important formalism in physics, mathematics, engineering, ...
  - **Informally:**
    - ♦ Sets of quantities, associated with point P (in n-dimensional space), which, upon change of coordinates, transform according to certain equations involving partial derivatives evaluated at P
  - Covariant, contravariant, mixed

Much simpler for  
subject matter of this course

# Tensors in CS and Deep Learning

- **Tensors in computer science (and in particular in deep-learning community)**
  - **Generalizations of 2D matrices to higher dimensions**
    - ♦ i.e., m-dimensional matrices, where m>2

Tensor → A

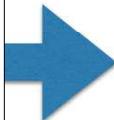
Tensor elements (components) →  $A_{ijk}$  or  $A_{i,j,k}$

$A_{ijklmn\dots}$  or  $A_{i,j,k,l,m,n,\dots}$

- In TensorFlow (discussed next)
  - ♦ Tensors (objects) are fundamental data-structures
    - *Typed multidimensional arrays*

## This Lecture

- Tensors in Deep Learning



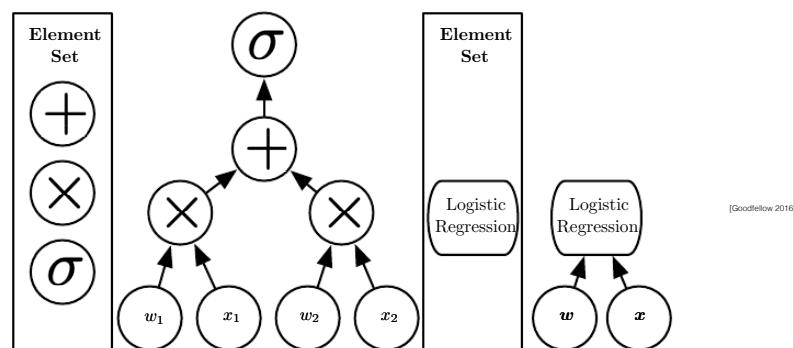
- Computational Graphs I

# Computational Graph

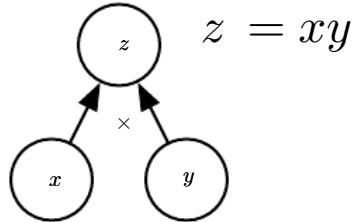
- **Graph  $G(V,E)$** 
  - $V$  — set of vertices (nodes)
  - $E$  — set of edges (connections between vertices)
- **Directed vs. undirected graphs**
- **Computational graph**
  - Directed graph
  - Nodes may represent
    - Operations
    - Structures, e.g., variables
    - Structures *and* operations

# Computational Graphs

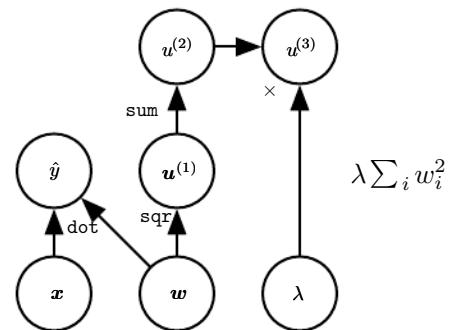
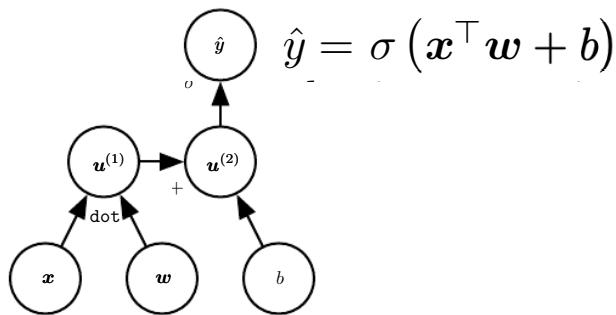
- Consider computational graph with nodes representing mathematical operations
  - Nodes may be at different levels of “complexity”
  - Example:



# Computational Graph Examples

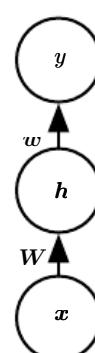
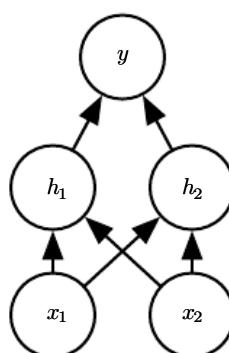
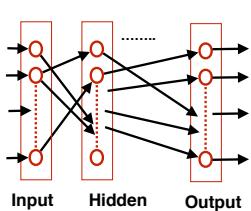


[Goodfellow, 2016]



9

## Feedforward ANN Graphical Representation



Computational graphs are of key importance to modern neural networks and deep learning  
• Including efficient training of deep networks

We will learn much more about computational graphs in upcoming lectures SOON (please be patient)

# **Questions?**

---

# **Neural Networks and Deep Learning**

**Dr. Jerome J. Braun**

## **This Lecture: Artificial Neural Networks — Fundamentals II**

Course: Neural Networks and Deep Learning  
IE 7615

---

---

**Unauthorized distributing, redistributing, posting,  
and/or reposting, of the materials of this course  
(including course lectures, lecture slides, slide-sets,  
syllabi, guidelines, assignments, quizzes, exams,  
presentations, software, electronic media, etc.),  
is prohibited.**

# Notation

---

- Note that in the following slides  $w$  and  $\theta$  are used interchangeably (i.e., synonymous)

# This Lecture

---

- ANN basics — continued
- Forward pass
- Cost functions

# Perceptron Learning

- Perceptron criterion

$$E_P(\mathbf{w}) = - \sum_{n \in M} \mathbf{w}^T \phi_n t_n$$

Note: M is of samples that were *misclassified*

- Perceptron learning rule

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_P(\mathbf{w}) = \mathbf{w}^{(\tau)} + \eta \phi_n t_n$$

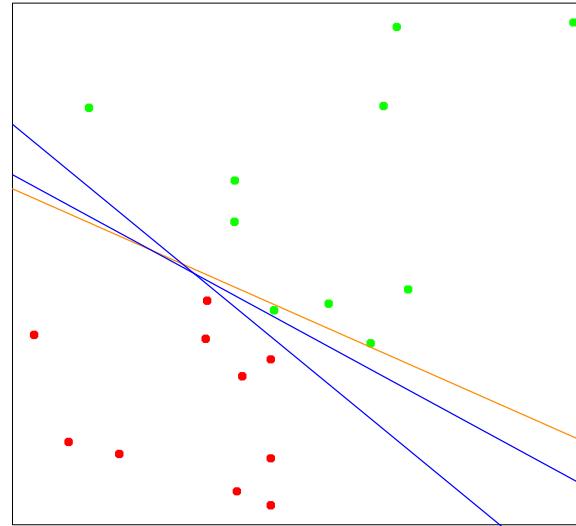
Learning-rate hyperparameter

# Perceptron Convergence Theorem

- Given
  - Set of P input patterns  $x(p)$ ,  $p=1, \dots, P$
  - Set of target values  $t(p)$ ,  $p=1, \dots, P$
  - Perceptron learning rule (wolog can set learning rate to 1)
- If
  - There is weight vector  $w^*$ , such that for all patterns  $p$ ,  $f(x(p)w^*) = t(p)$
- Then
  - For any starting vector  $w$ , perceptron learning algorithm will converge to weight vector that gives correct response for all training patterns in finite number of weight-update steps

# Perceptron on Separable Data

- Toy problem — separable by hyperplane
- Least squares solution misclassifies one point
  - Orange line
- Perceptron
  - Two different random initializations
  - Blue lines



J. Braun

7

# Perceptron Limitations

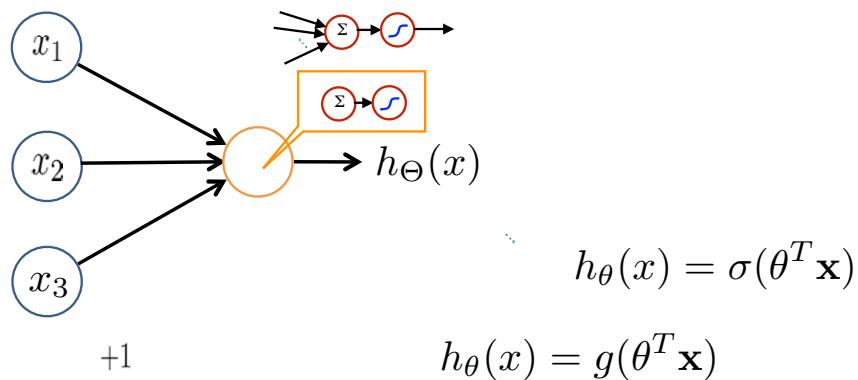
- Convergence — finite but number of updates may be large
- Perceptron cannot cope with simple problems
  - XOR

J. Braun

8

# Single Layer

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$



# Activation Functions

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

- **Sigmoid**
- **Hyperbolic tangent**  $f'(x) = m f(x)[1 - f(x)]$
- **Rectified linear unit (ReLU)**
- **And others**  $h(a) \equiv \tanh(a)$

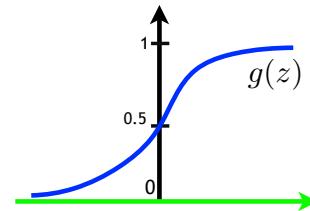
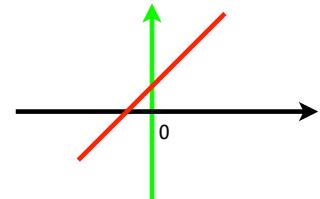
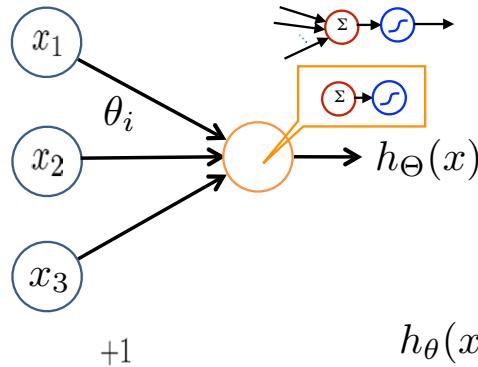
$$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}.$$

$$h'(a) = 1 - h(a)^2$$

# Single Layer

$$h_{\theta}(x) = \theta^T \mathbf{x}$$

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$



$$y = \sigma(x) = g(x) = \frac{1}{1 + \exp(-cx)}$$

$$g(x) = \frac{1}{1 + \exp(-x)}$$

$$h_{\theta}(x) = g(\theta^T \mathbf{x})$$

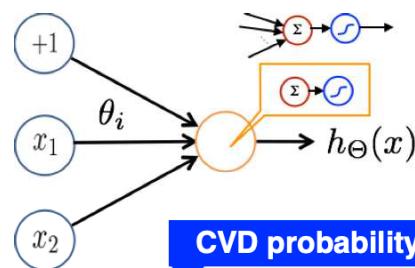
J. Braun

11

# Toy Illustration

- Predict heart disease risk (CVD) from lipid panel (“cholesterol blood-test”) data

$$\mathbf{x} = \begin{bmatrix} +1 \\ \text{LDL [mg/dL]} \\ \text{HDL [mg/dL]} \end{bmatrix}$$

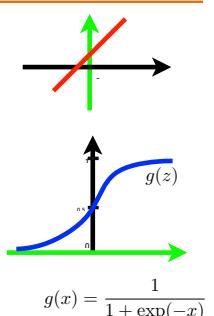


**CVD probability (i.e., that CVD=1)**

$$h_{\theta}(\mathbf{x}) = 0.65$$

$$p(\text{CVD}=0|\mathbf{x}; \theta) + p(\text{CVD}=1|\mathbf{x}; \theta) = 1$$

$$p(\text{CVD}=0|\mathbf{x}; \theta) = 1 - p(\text{CVD}=1|\mathbf{x}; \theta)$$



$$g(x) = \frac{1}{1 + \exp(-x)}$$

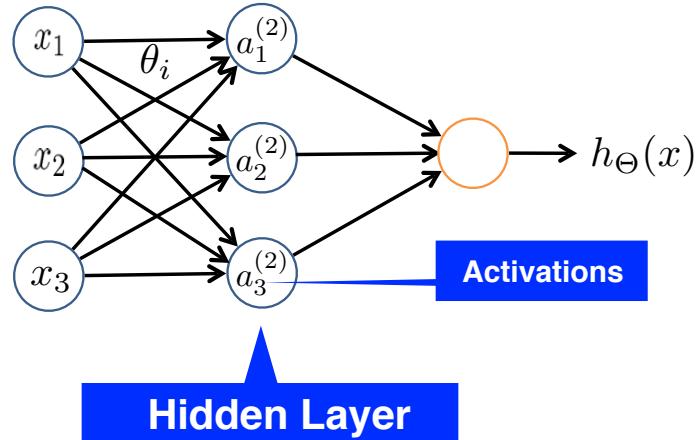
• For illustration only !  
• NOT suitable for any practical use !

J. Braun

12

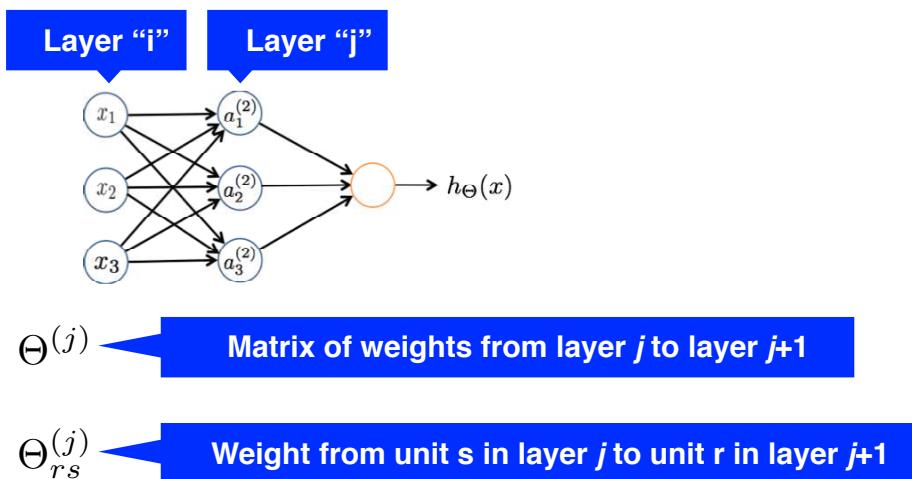
# Adding Layers

Also bias in each layer



J. Braun

## Adding Layers – Weights Matrix



J. Braun

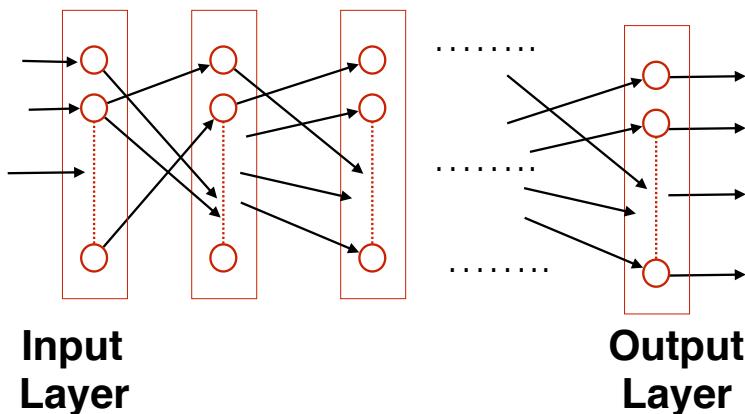
# Network Depth

- Universal Approximation Theorem
  - One hidden layer sufficient to approximate ANY function  $F$  to ARBITRARY degree of accuracy
- But:
  - Universal approximation theorem does NOT imply ability to LEARN the function, i.e., given any function  $F$ 
    - Single hidden-layer network approximator which represents any desired function  $F$  **does** exist
    - But learning algorithm may not be able to learn (i.e., find) it
    - Problem of hidden-layer size (width)

Rationale  
for  
Deep  
Learning

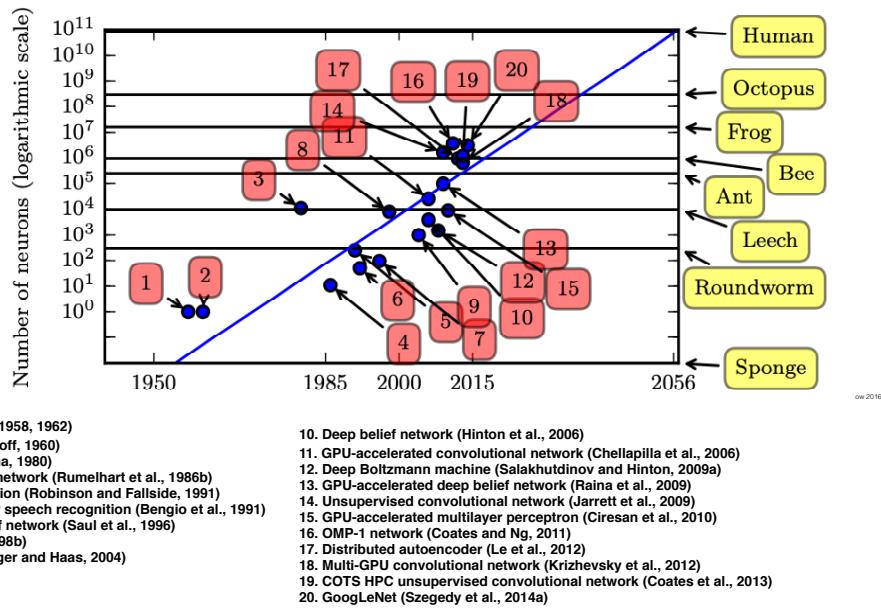
## Deep Feedforward MLP Architectures (Recap)

### Multiple Hidden Layers



## Artificial vs. Biological Neural Networks Number of Units/Neurons

**ANNs and projection c.2015**

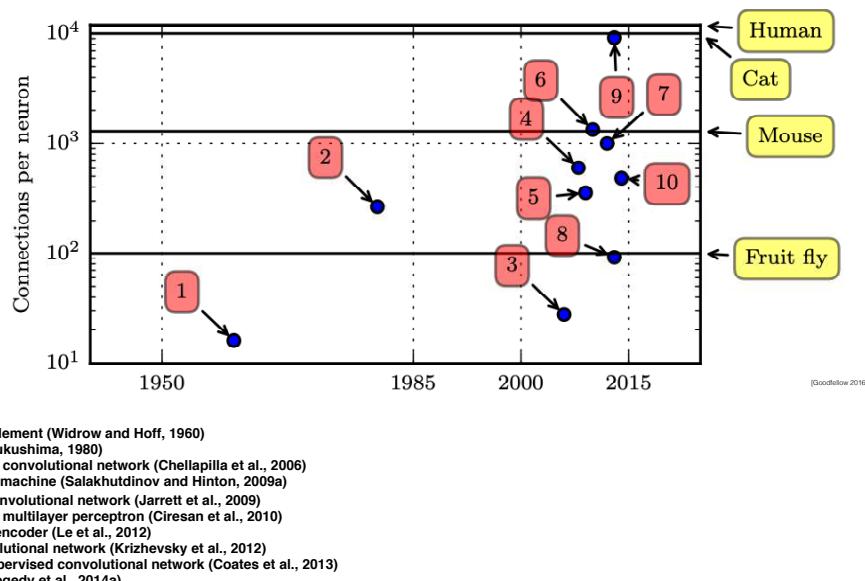


Basics of biological neural networks (and contrasts with ANNs) will be discussed in later lecture

17

## Connections per Unit/Neuron

**ANNs and projection c.2015**

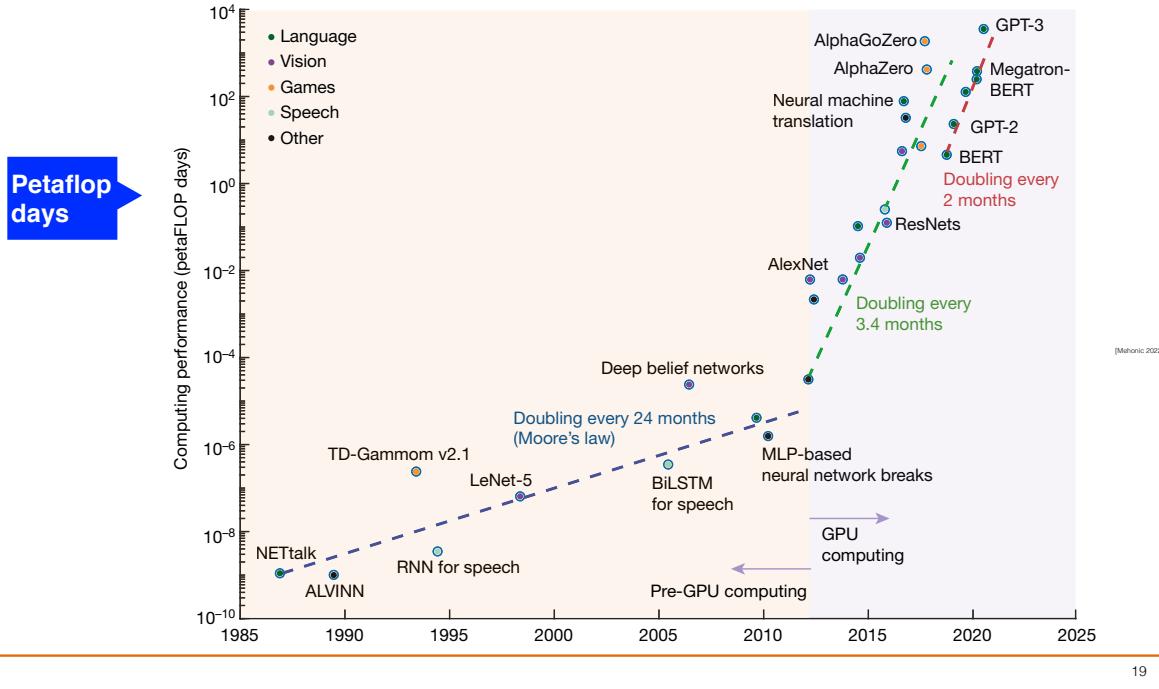


Basics of biological neural networks (and contrasts with ANNs) will be discussed in later lecture

18

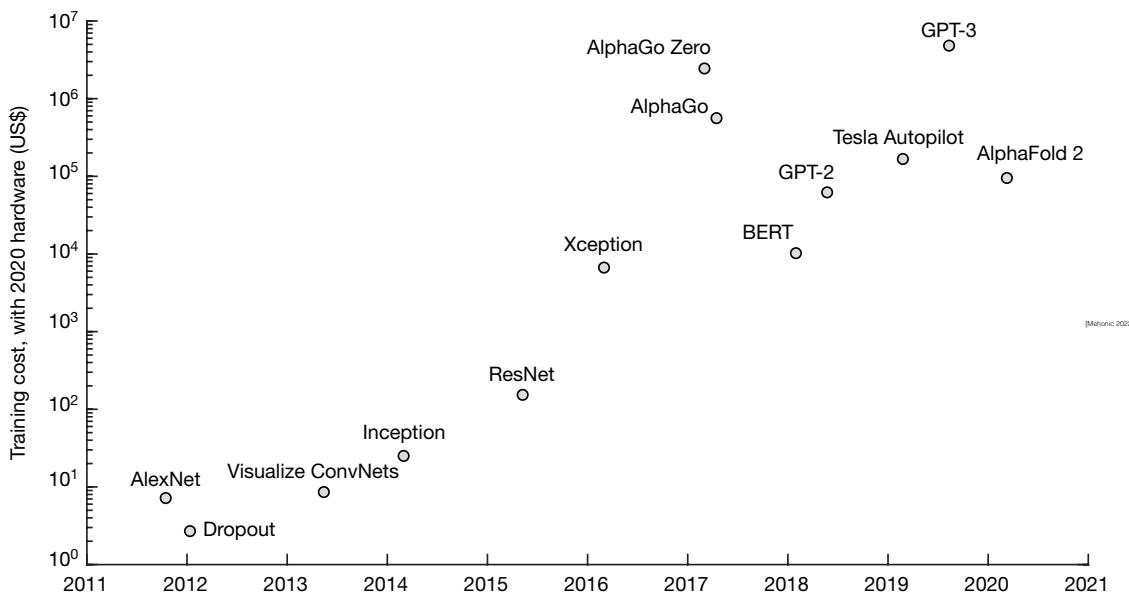
## ANN Computational Demands – “Compute Power”

- Mehonic & Kenyon (Nature, 2022)

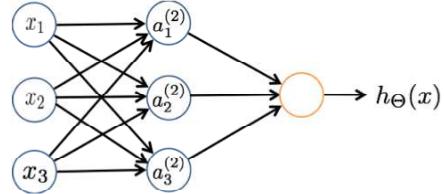


## ANN Training Cost with Year-2020 Hardware

- Mehonic & Kenyon (Nature, 2022)



# Forward Pass



$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$

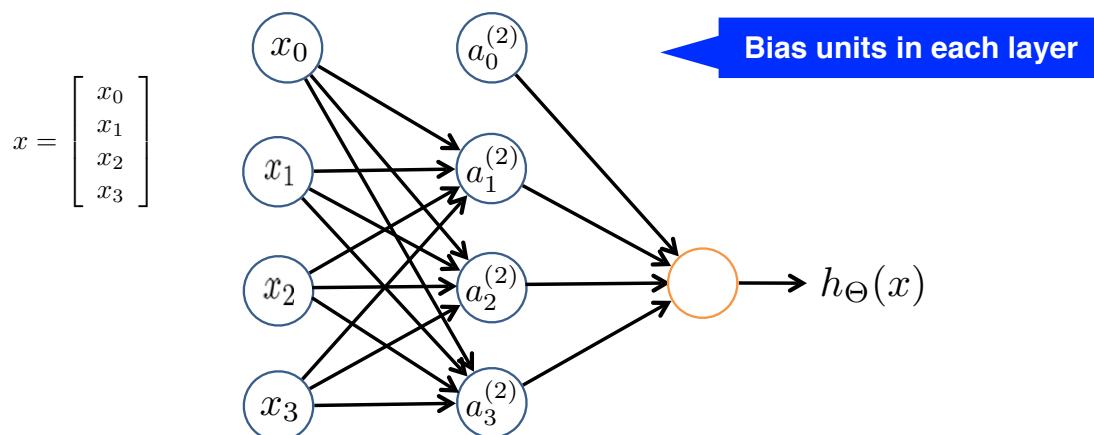
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$

$$a^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

$$h_\theta(x) = a^{(3)}$$

J. Braun

# Forward Pass



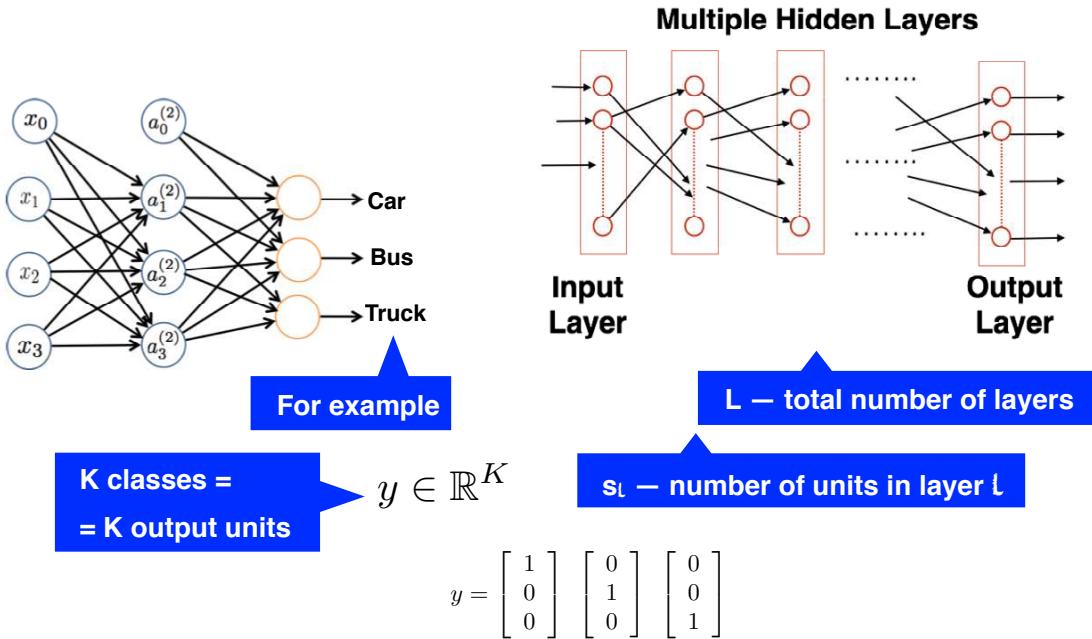
Suitable for  
vectorization

$$a^{(2)} = g(\Theta^{(1)} x)$$

$$a^{(3)} = g(\Theta^{(2)} a^{(2)}) = h_\theta(x)$$

J. Braun

# Multiclass Classification



J. Braun

# Cost Functions

- Regularized logistic regression

$$E(\theta) = -\frac{1}{m} \left[ \sum_{i=1}^m y^{(i)} \ln h_\theta(x^{(i)}) + (1 - y^{(i)}) \ln (1 - h_\theta(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

**m exemplars**      **Training error**      **Regularization**

- Feedforward MLP network

$$\text{K classes} \rightarrow y \in \mathbb{R}^K$$

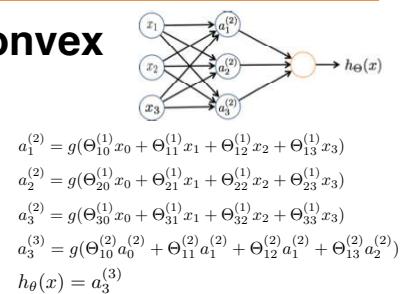
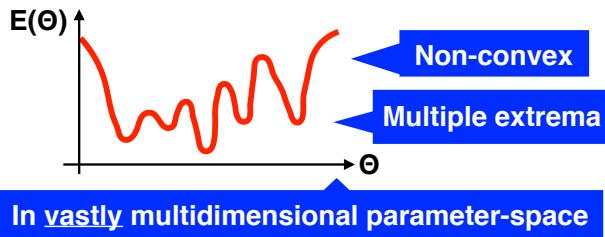
$$E(\Theta) = -\frac{1}{m} \left[ \sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \ln(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \ln(1 - (h_\Theta(x^{(i)}))_k) \right] + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2$$

**Training error**      **Regularization**

J. Braun

# Error Function Landscape

- For ANNs cost functions are non-convex



$$E(\Theta) = -\frac{1}{m} \left[ \sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \ln(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \ln(1 - (h_\Theta(x^{(i)}))_k) \right] + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2$$

Need  $\text{argmin}_\Theta E(\Theta)$  Will cover in upcoming lectures

## Questions?

---

# **Neural Networks and Deep Learning**

**Dr. Jerome J. Braun**

## **This Lecture: Artificial Neural Networks - Fundamentals III**

Course: Neural Networks and Deep Learning  
IE 7615

---

---

**Unauthorized distributing, redistributing, posting,  
and/or reposting, of the materials of this course  
(including course lectures, lecture slides, slide-sets,  
syllabi, guidelines, assignments, quizzes, exams,  
presentations, software, electronic media, etc.),  
is prohibited.**

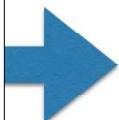
# Notation

---

- Note that in the following slides  $w$  and  $\theta$  are used interchangeably (i.e., synonymous)

# This Lecture

---



- Toy example
- Feedforward MLP network fundamentals
- Feedforward MLP network training basics
  - Basic back-propagation in feedforward MLP networks

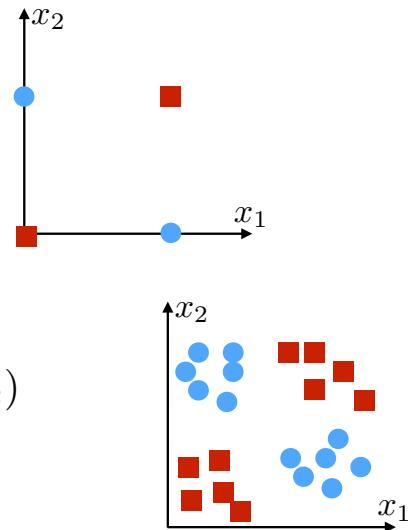
## Example: XOR/XNOR

- Non-linear decision boundary example
- Binary x values

$$x_1, x_2 \in \{0, 1\}$$

$$y = x_1 \text{ XOR } x_2$$

$$x_1 \text{ XNOR } x_2 \leftrightarrow \text{NOT}(x_1 \text{ XOR } x_2)$$



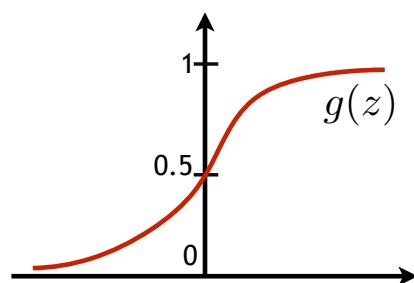
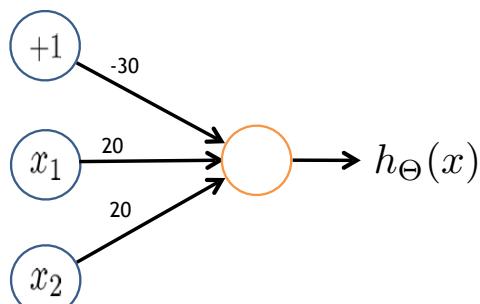
J. Braun

5

## Example: AND

$$x_1, x_2 \in \{0, 1\}$$

$$x_1 \text{ AND } x_2$$



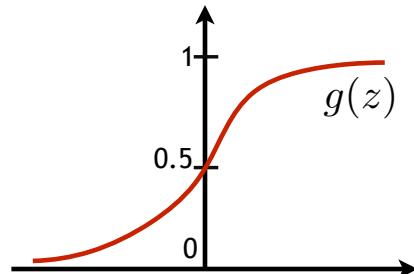
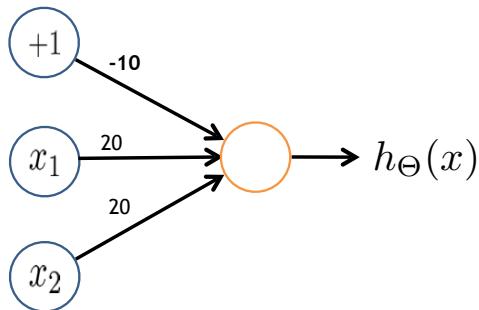
$x_1$	$x_2$	$h$
0	0	0
0	1	0
1	0	0
1	1	1

6

## Example: OR

$$x_1, x_2 \in \{0, 1\}$$

$x_1$  OR  $x_2$

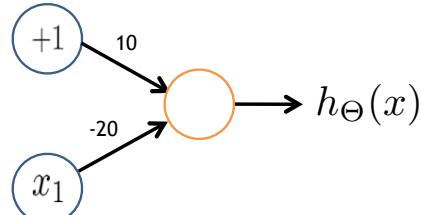


$x_1$	$x_2$	$h$
0	0	0
0	1	1
1	0	1
1	1	1

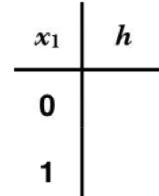
7

## Example: NOT

- Negation



$$h_{\Theta}(x) = g(10 - 20x_1)$$

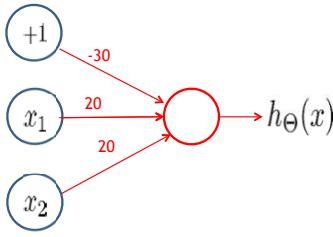


- Combine NOT, AND, OR, for e.g.:

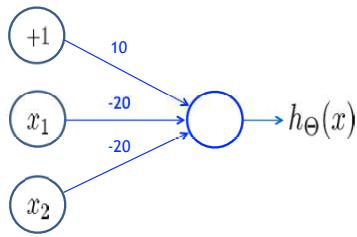
$$(\text{NOT } x_1) \text{ AND } (\text{NOT } x_2)$$

8

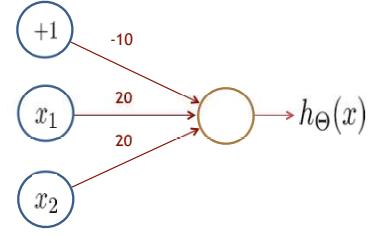
# XNOR



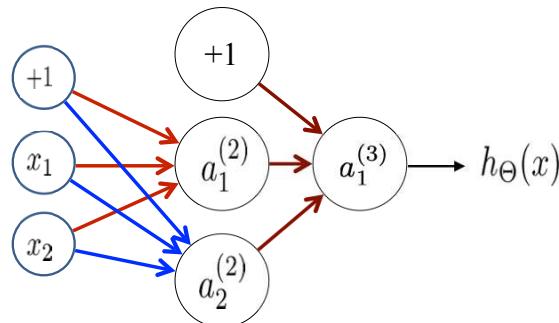
$x_1 \text{AND } x_2$



(NOT  $x_1$ ) AND (NOT  $x_2$ )



$x_1 \text{OR } x_2$



$x_1$	$x_2$	$a_1^{(2)}$	$a_2^{(2)}$	$h$
0	0	-25	15	0
0	1	-25	15	0
1	0	15	-25	0
1	1	15	-25	1

9

## XNOR – Different Weight Parameters

$$\begin{bmatrix} -35 \\ 20 \\ 20 \end{bmatrix}$$

$$\begin{bmatrix} 15 \\ -25 \\ -25 \end{bmatrix}$$

$$\begin{bmatrix} -15 \\ 25 \\ 25 \end{bmatrix}$$

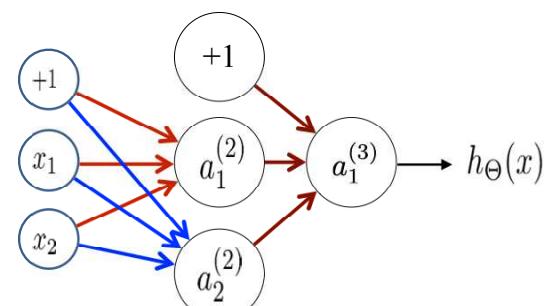
$$\begin{bmatrix} 15 \\ -25 \end{bmatrix}$$

$$\begin{bmatrix} -25 \\ 15 \\ 15 \end{bmatrix}$$

$$\begin{bmatrix} -5 \\ 15 \\ 15 \end{bmatrix}$$

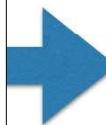
$$\begin{bmatrix} 5 \\ -15 \\ -15 \end{bmatrix}$$

$$\begin{bmatrix} 5 \\ -15 \end{bmatrix}$$



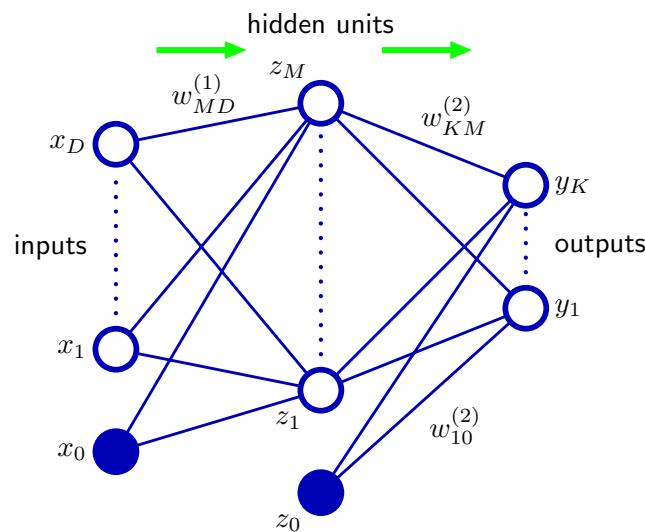
$x_1$	$x_2$	$a_1^{(2)}$	$a_2^{(2)}$	$h$
0	0	-25	15	0
0	1	-25	15	0
1	0	15	-25	0
1	1	15	-25	1

# This Lecture



- Toy example
- Feedforward MLP network fundamentals
- Feedforward MLP network training basics
  - Basic back-propagation in feedforward MLP networks

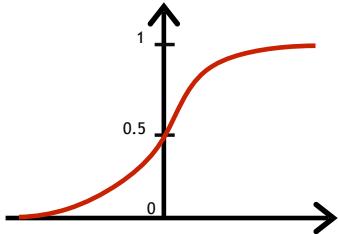
## Feedforward ANN with One Hidden Layer



## Recap: Sigmoidal Squashing Functions

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

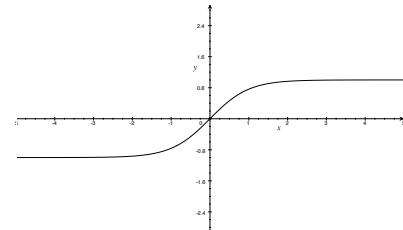
$$f'(x) = mf(x)[1 - f(x)]$$



$$h(a) \equiv \tanh(a)$$

$$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}.$$

$$h'(a) = 1 - h(a)^2$$

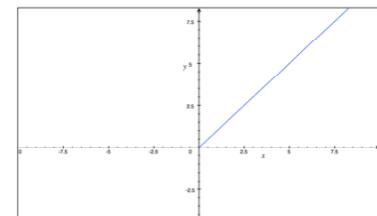


J. Braun

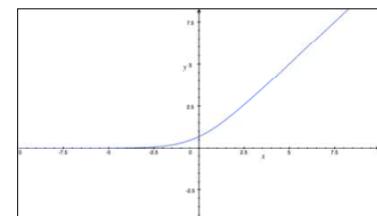
13

## ReLU – Rectified Linear Unit

- $f(x) = \max(0, x)$ 
  - where  $x$  is input to activation function
- Avoids saturation issue
- Mitigate vanishing gradient problem
- “Default recommendation” for modern feedforward ANNs (Goodfellow, 2016)



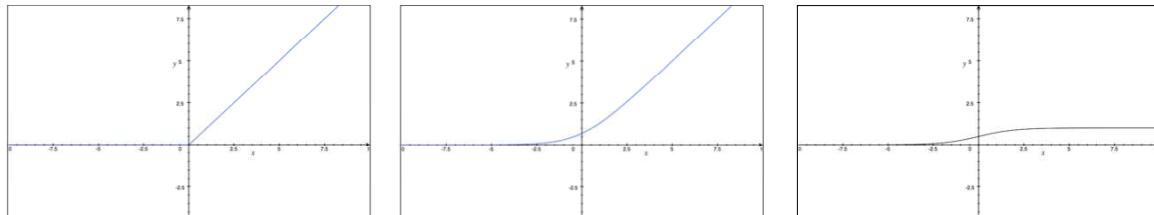
- Smooth approximation
  - Softplus
    - $f(x) = \log(1 + \exp(x))$
  - Derivative of softplus is sigmoid



J. Braun

14

# ReLU and Softplus vs. Sigmoid



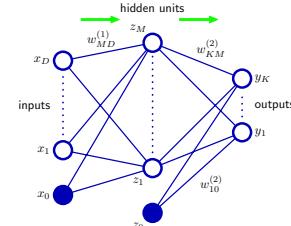
- Vanishing gradient

# ANN Parameters

- Representation of linear models for regression or classification
  - Linear combinations of basic functions  $\phi_j(\mathbf{x})$

$$y(\mathbf{x}, \mathbf{w}) = f \left( \sum_{j=1}^M w_j \phi_j(\mathbf{x}) \right)$$

- Basis functions  $\phi_j(\mathbf{x})$  in ANN
  - Depend on parameters
  - Parameters of basis functions adapted during training along with weights  $\{w_j\}$



# Network Training – SSE

---

- Consider ANN with D inputs and M hidden units
  - With sigmoid activation function we have:

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma \left( \sum_{j=0}^M w_{kj}^{(2)} h \left( \sum_{i=0}^D w_{ji}^{(1)} x_i \right) \right).$$

- By analogy with polynomial curve fitting

- For inputs  $\mathbf{x}_n$  and target outputs  $\mathbf{t}_n$  where  $n=1\dots N$ , can minimize SSE

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \| \mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n \|^2$$

- More general view of network training possible giving network outputs a probabilistic interpretation

## Probabilistic View: ANN and Regression Task

---

- One real-value target output variable
  - Gaussian distributed with x-dependent mean (restrictive assumption!)

$$p(t|\mathbf{x}, \mathbf{w}) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1})$$

- Likelihood function:

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^N p(t_n|\mathbf{x}_n, \mathbf{w}, \beta)$$

- Taking negative logarithm get error function:

$$\frac{\beta}{2} \sum_{n=1}^N \{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 - \frac{N}{2} \ln \beta + \frac{N}{2} \ln(2\pi)$$

- can use this to learn parameters  $w$  and  $\beta$

# Minimizing Error Function

- Minimizing error function  $\leftrightarrow$  maximizing (log) likelihood
  - Minimize  $E(\mathbf{w})$ 
    - Yields maximum-likelihood solution  $\mathbf{w}_{ML}$
    - In ANNs,  $E(\mathbf{w})$  is non-convex
  - Once is  $\mathbf{w}_{ML}$  is found
    - Can find  $\beta_{ML}$  and target distribution given input
  - Similarly for multiple target variables
- $$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2$$
- $$\frac{1}{\beta_{ML}} = \frac{1}{N} \sum_{n=1}^N \{y(\mathbf{x}_n, \mathbf{w}_{ML}) - t_n\}^2$$
- $$p(t|\mathbf{x}, \mathbf{w}) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1})$$
- $$\frac{1}{\beta_{ML}} = \frac{1}{NK} \sum_{n=1}^{NK} \|y(\mathbf{x}_n, \mathbf{w}_{ML}) - \mathbf{t}_n\|^2$$
- $$p(\mathbf{t}|\mathbf{x}, \mathbf{w}) = \mathcal{N}(\mathbf{t}|y(\mathbf{x}, \mathbf{w}), \beta^{-1}\mathbf{I})$$

J. Braun

19

# Binary Classification

- Single target variable  $t$  where
    - class  $C_0$  if  $t=0$ , class  $C_1$  if  $t=1$
    - logistic sigmoid activation function
  - Then conditional target distribution given inputs
    - Bernoulli form
  - For training set of independent observations, error function (given by negative log likelihood) is
    - Cross-entropy error function
  - Using cross-entropy error function instead of SSE
    - Faster training, improved generalization (Simard et al., 2003)
- $$y = \sigma(a) \equiv \frac{1}{1 + \exp(-a)}$$
- $$p(t|\mathbf{x}, \mathbf{w}) = y(\mathbf{x}, \mathbf{w})^t \{1 - y(\mathbf{x}, \mathbf{w})\}^{1-t}$$
- $$E(\mathbf{w}) = - \sum_{n=1}^N \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$$
- where  $y_n$  denotes  $y(\mathbf{x}_n, \mathbf{w})$

J. Braun

20

# K Separate Binary Classifications

- **K outputs**

- target variable  $t_k$  associated with each  $t_k \in \{0, 1\}$ , where  $k = 1, \dots, K$
  - logistic sigmoid activation functions

- **Assuming class labels independent given input vector**

- Conditional distribution of targets is:

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}) = \prod_{k=1}^K y_k(\mathbf{x}, \mathbf{w})^{t_k} [1 - y_k(\mathbf{x}, \mathbf{w})]^{1-t_k}$$

- **Taking negative log of corresponding likelihood function, find  $E(\mathbf{w})$ :**

- $E(\mathbf{w}) = - \sum_{n=1}^N \sum_{k=1}^K \{t_{nk} \ln y_{nk} + (1 - t_{nk}) \ln(1 - y_{nk})\}$   
where  $y_{nk}$  denotes  $y_k(\mathbf{x}_n, \mathbf{w})$

# Multiclass Classification

- **K mutually exclusive classes**

- $t_k$  have 1-of-K coding scheme  $t_k \in \{0, 1\}$

- **Network outputs interpreted as:**

$$y_k(\mathbf{x}, \mathbf{w}) = p(t_k = 1 | \mathbf{x})$$

- **Leads to error function:**  $E(\mathbf{w}) = - \sum_{n=1}^N \sum_{k=1}^K t_{kn} \ln y_k(\mathbf{x}_n, \mathbf{w})$

$$a_k = \mathbf{w}_k^T \phi$$

- **Output activation given by softmax function** 

- (Ref. Goodfellow textbook section 6.2.2.3)

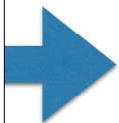
$$y_k(\mathbf{x}, \mathbf{w}) = \frac{\exp(a_k(\mathbf{x}, \mathbf{w}))}{\sum_j \exp(a_j(\mathbf{x}, \mathbf{w}))}$$

which satisfies  $0 \leq y_k \leq 1$  and  $\sum_k y_k = 1$

# This Lecture

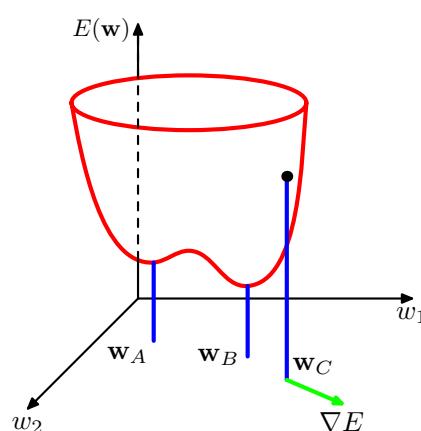
---

- Toy example
- Feedforward MLP network fundamentals
- Feedforward MLP network training basics
  - Basic back-propagation in feedforward MLP networks



## Geometrical View of Error Function

---



- Error function  $E(\mathbf{w})$  as surface sitting over weight space
- Point  $\mathbf{w}_A$  is **local** minimum
- Point  $\mathbf{w}_B$  is **global** minimum
- At any point  $\mathbf{w}_C$ , local gradient of error surface is given by vector  $\nabla E$

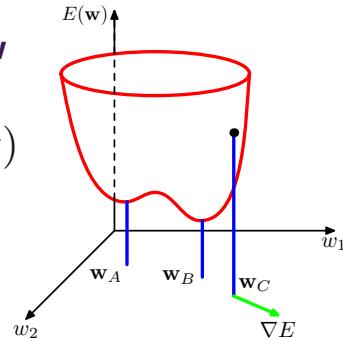
# Parameter Optimization

- Take small step in weight space from  $w$  to  $w + \delta w$

- Change in error function:

$$\delta E \simeq \delta \mathbf{w}^T \nabla E(\mathbf{w})$$

- where the vector  $\nabla E(w)$  points in direction of greatest rate of increase of error function



- Assuming continuity, minimum when:  $\nabla E(\mathbf{w}) = 0$

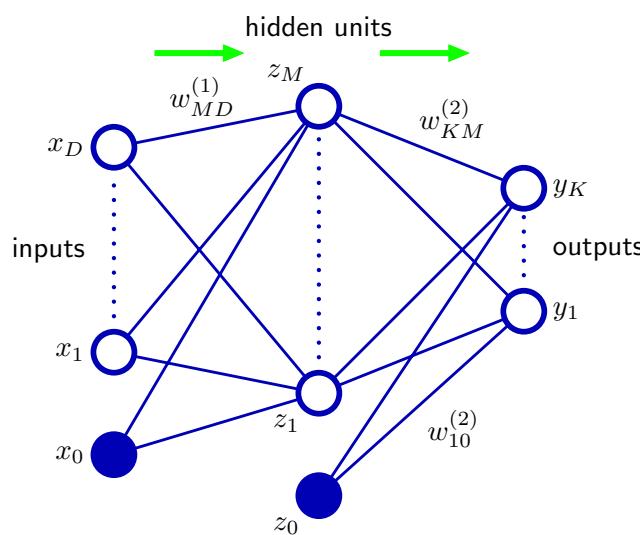
- Many equivalent minima and inequivalent local minima and stationary points

- No hope for finding analytical solution

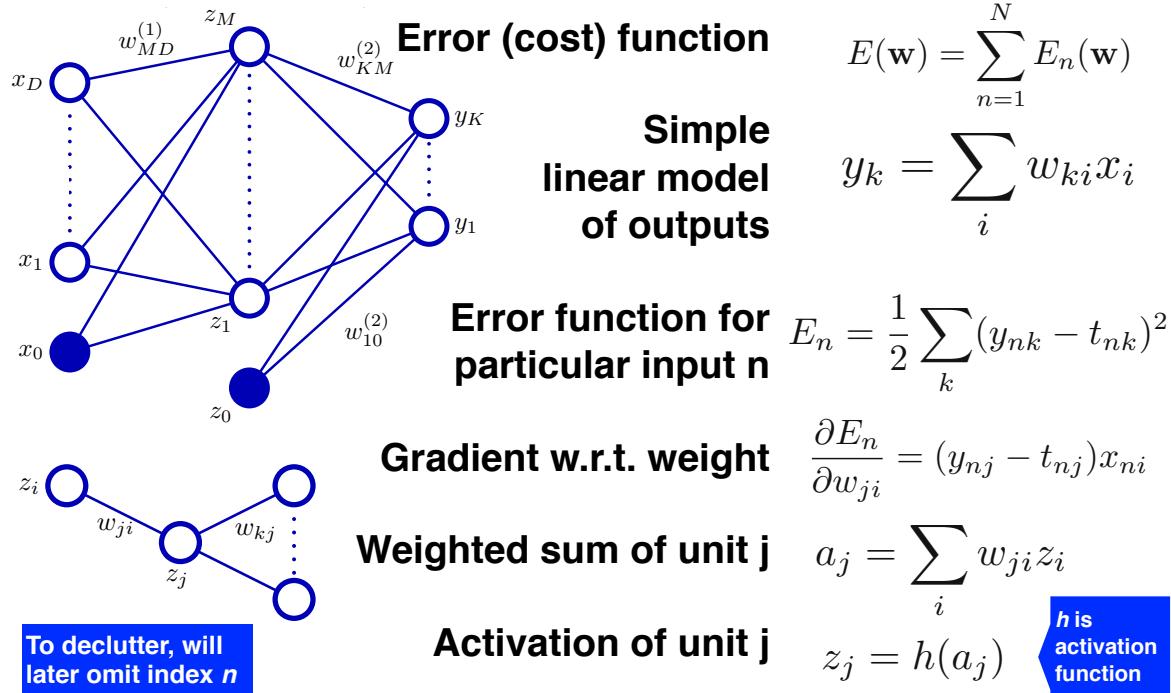
- Need iterative solutions

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta \mathbf{w}^{(\tau)}$$

# Training Feedforward ANN



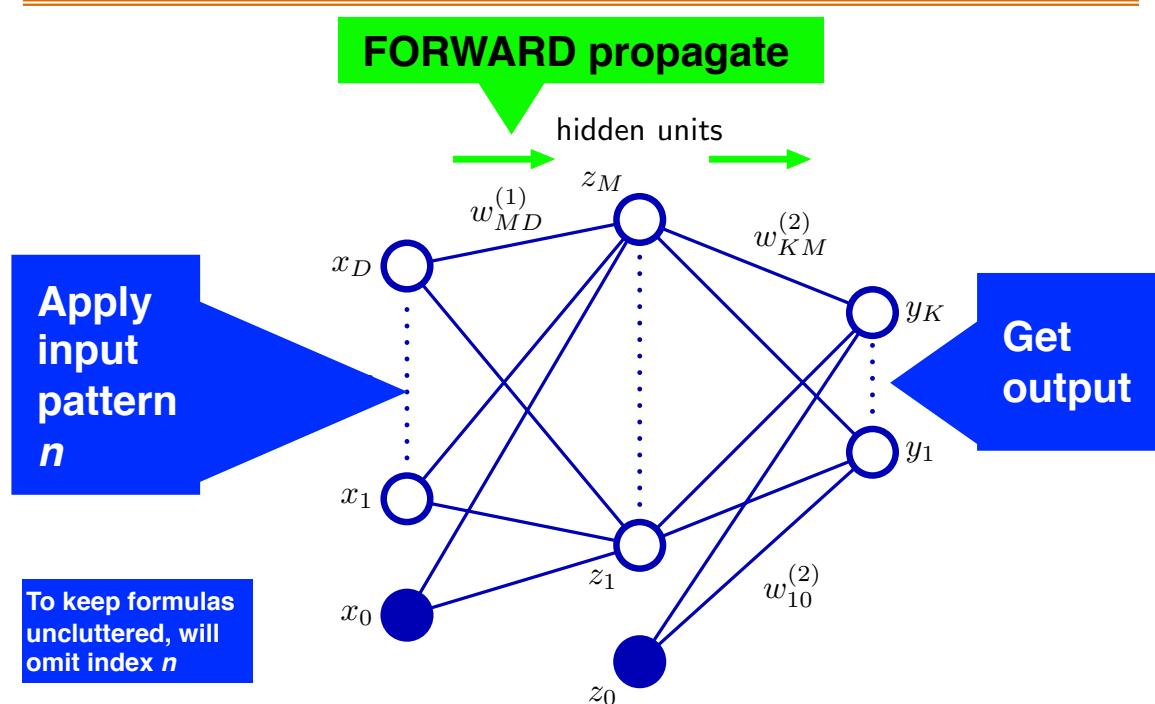
# Basic Back-propagation



J. Braun

27

## Basic Back-propagation: Forward Pass



J. Braun

28

## Basic Back-propagation – Error Signals

- $E_n$  depends on weight  $w_{ji}$  only via summed input  $a_j$  to unit  $j$ 
  - Therefore can apply chain rule

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}$$

Define error  
( a.k.a. error signal )

Denote errors as  $\delta$

Note: Errors (a.k.a. error signals)  $\delta$  should NOT be confused with error-function (a.g.a. cost, loss)  $E_n$

Recalling ➤  $a_j = \sum_i w_{ji} z_i$

Yields ➤

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$$

J. Braun

29

## Basic Back-propagation – Errors at Output Units

Recall, that for  
OUTPUT layer

$$E_n = \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2$$

Recall, that for  
OUTPUT layer

$$\frac{\partial a_j}{\partial w_{ji}} = z_i$$

Error signals for  
OUTPUT layer units  $k$

$$\delta_k = y_k - t_k$$

Ex.

Easy to compute errors at  
output, because at outputs  
we have values of targets  $t_k$

J. Braun

30

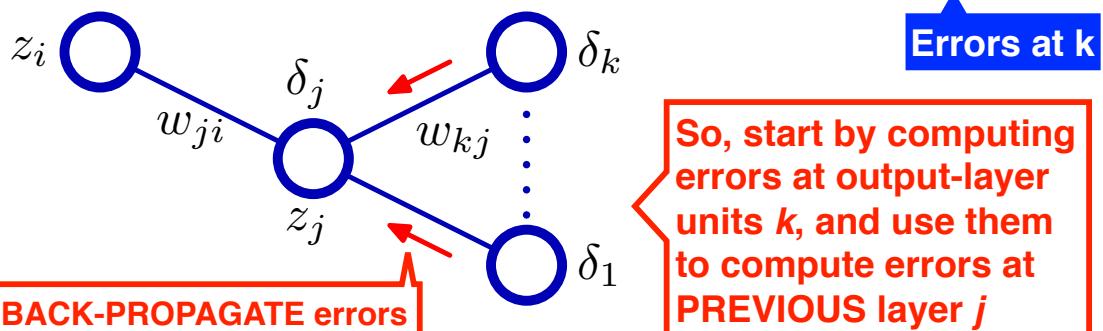
## Back-propagating Errors – Errors at Hidden Layer Units

It was easy to compute error signals at output  $\delta_k = y_k - t_k$

But in hidden-layers we do not have target values !

However, by using chain rule,  
we can relate errors at hidden  
layer  $j$  to errors at NEXT layer  $k$

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j}$$



J. Braun

31

## Errors at Hidden Layer Units (cont.)

Recall

$$a_j = \sum_i w_{ji} z_i$$

$$z_j = h(a_j)$$

Activation function

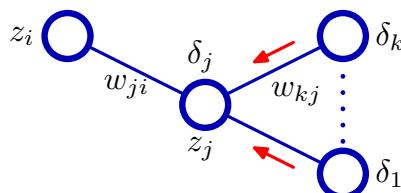
Using the above, and  
substituting definition of  $\delta_k$   
into sum in expression for  $\delta_j$

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j}$$

After substitution, we get

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$$

Ex.



J. Braun

32

# Basic Back-propagation Steps Summary

- Apply input vector  $x_n$  to network and **forward-propagate** through network using **(B1)** and **(B2)** to find activations of all hidden units and output units
- Evaluate errors  $\delta_k$  for all output units using **(B3)**
- **Back-propagate error-signals** ( $\delta$ 's) using **(B4)** to obtain  $\delta_j$  for each hidden unit
- Use **(B5)** to evaluate derivatives of cost-function

$$(B1) \quad a_j = \sum_i w_{ji} z_i$$

$$(B2) \quad z_j = h(a_j)$$

$$(B3) \quad \delta_k = y_k - t_k$$

$$(B4) \quad \delta_j = h'(a_j) \sum_k w_{kj} \delta_k$$

$$(B5) \quad \frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$$

- Up to now:
  - Covered foundations of basic back-propagation
- In upcoming lectures:
  - Will cover modern back-propagation techniques, exploiting computational graphs, currently used in training of deep neural networks

# Questions?

---

---

# **Neural Networks and Deep Learning**

**Dr. Jerome J. Braun**

## **This Lecture: ANN Fundamentals IV, Jacobian and Hessian**

Course: Neural Networks and Deep Learning  
IE 7615

---

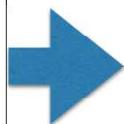
J. Braun

---

**Unauthorized distributing, redistributing, posting,  
and/or reposting, of the materials of this course  
(including course lectures, lecture slides, slide-sets,  
syllabi, guidelines, assignments, quizzes, exams,  
presentations, software, electronic media, etc.),  
is prohibited.**

# This Lecture

---



- Basic back-propagation example
- Jacobian and Hessian

# Notation

---

$w$  or  $\theta$  Weights, network parameters

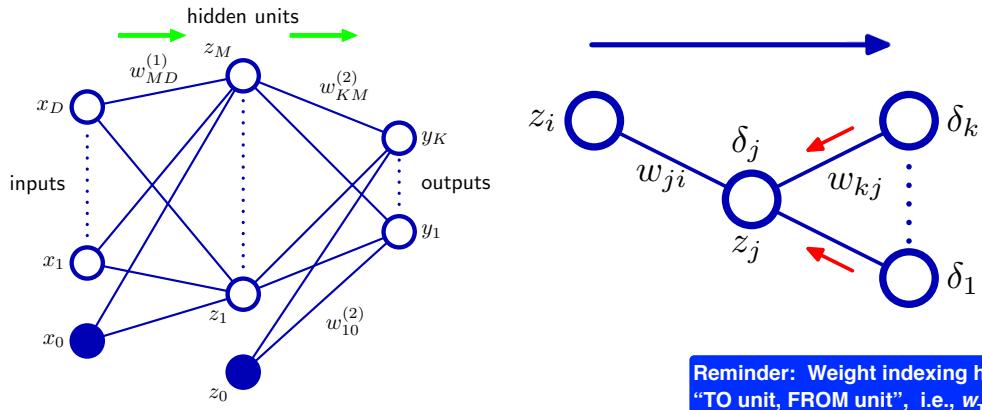
$\epsilon$  or  $\eta$  Learning rate

$g$  or  $b$  Gradient

$E(w)$  or  $J(w)$  Error function, cost function

$J$  Jacobian

# Basic Back-propagation Recap



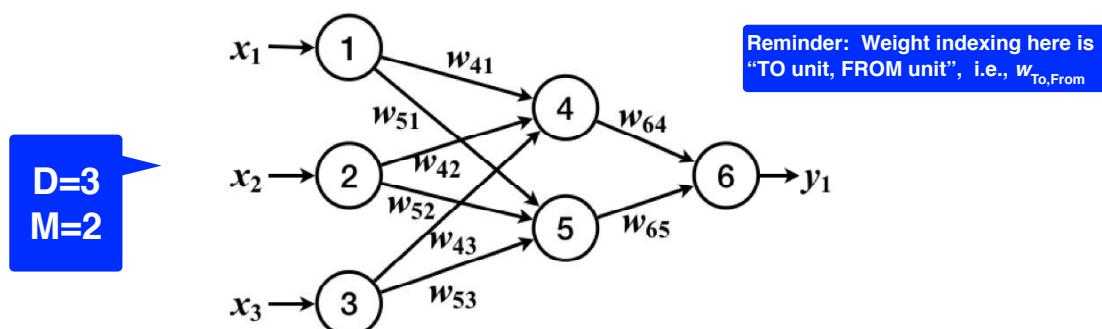
- Recall detailed derivation of basic back-propagation formulas (as we did in previous lecture)

Now apply those formulas to simple numerical example

J. Braun

5

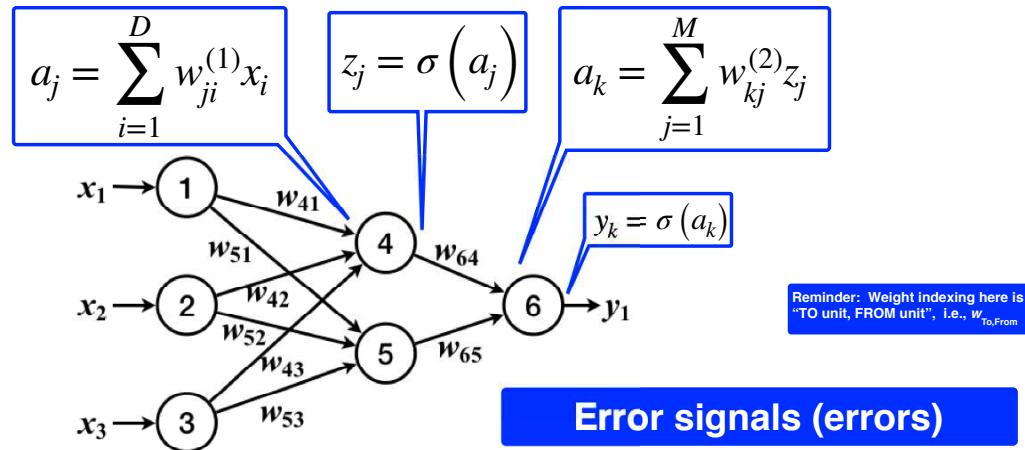
# Basic Back-propagation Example



- Binary input vectors
- Sigmoid activation function for hidden and output layers
- Assume learning rate  $\eta=0.9$
- Consider single training example  $[1,0,1]^T$ 
  - Class label  $t=1$

6

## Basic Back-propagation Example (cont. 2)



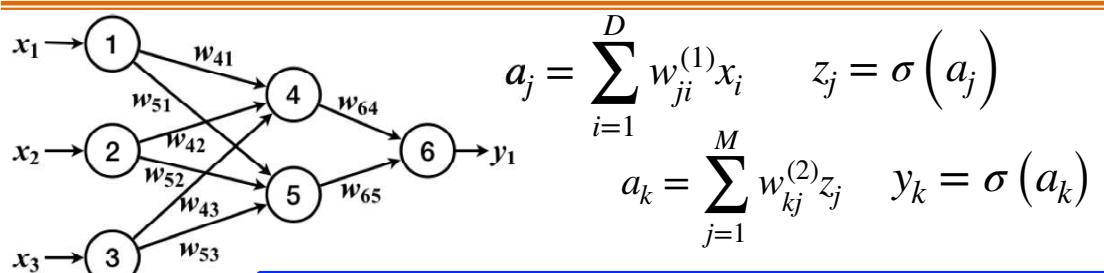
### Cost function derivatives

$$\delta_k = \sigma'(a_k)(y_k - t_k)$$

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i \quad \frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j \quad \delta_j = \sigma'(a_j) \sum_k w_{kj} \delta_k$$

7

## Basic Back-propagation Example (cont. 3): Activation Sums and Outputs of Units



Suppose current weights at training iteration  $\tau$  are:

X <sub>1</sub>	X <sub>2</sub>	X <sub>3</sub>
1	0	1

W <sub>41</sub>	W <sub>51</sub>	W <sub>42</sub>	W <sub>52</sub>	W <sub>43</sub>	W <sub>53</sub>	W <sub>64</sub>	W <sub>65</sub>	W <sub>40</sub>	W <sub>50</sub>	W <sub>60</sub>
0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	-0.4	0.2	0.1

Then:

Unit	Activation sum $a$	Unit output $\sigma(a)$
4	$1(0.2) + 0(0.4) + 1(-0.5) + 1(-0.4) = -0.7$	$1/(1+e^{-0.7}) = 0.332$
5	$1(-0.3) + 0(0.1) + 1(0.2) + 1(0.2) = 0.1$	$1/(1+e^{-0.1}) = 0.525$
6	$(-0.3)(0.332) + (-0.2)(0.525) + 1(0.1) = -0.105$	$1/(1+e^{-0.105}) = 0.474$

8

## Basic Back-propagation Example (cont. 4): Error Signals

For output layer

$$\delta_k = \sigma'(a_k)(y_k - t_k) = [\sigma(a_k)(1 - \sigma(a_k))](1 - \sigma(a_k))$$

- Use  $(t_k - y_k)$
- Will then need plus-sign in weight-update equations

For hidden layer

$$\delta_j = \sigma'(a_j) \sum_k w_{kj} \delta_k = [\sigma(a_j)(1 - \sigma(a_j))] \sum_k w_{kj} \delta_k$$

9

## Basic Back-propagation Example (cont. 5): Error Signals

X <sub>1</sub>	X <sub>2</sub>	X <sub>3</sub>	W <sub>41</sub>	W <sub>51</sub>	W <sub>42</sub>	W <sub>52</sub>	W <sub>43</sub>	W <sub>53</sub>	W <sub>64</sub>	W <sub>65</sub>	W <sub>40</sub>	W <sub>50</sub>	W <sub>60</sub>
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	-0.4	0.2	0.1

Unit	Activation sum $a$	Unit output $\sigma(a)$
4	$1(0.2) + 0(0.4) + 1(-0.5) + 1(-0.4) = -0.7$	$1/(1+e^{-0.7}) = 0.332$
5	$1(-0.3) + 0(0.1) + 1(0.2) + 1(0.2) = 0.1$	$1/(1+e^{-0.1}) = 0.525$
6	$(-0.3)(0.332) + (-0.2)(0.525) + 1(0.1) = -0.105$	$1/(1+e^{0.105}) = 0.474$

- Use  $(t_k - y_k)$
- Will then use plus-sign in weight-update equations

$$\delta_k = \sigma'(a_k)(y_k - t_k) = [\sigma(a_k)(1 - \sigma(a_k))](1 - \sigma(a_k))$$

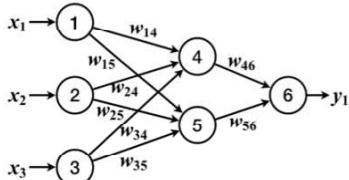
$$\delta_j = \sigma'(a_j) \sum_k w_{kj} \delta_k = [\sigma(a_j)(1 - \sigma(a_j))] \sum_k w_{kj} \delta_k$$

Therefore →

Unit	Error signal $\delta$
6	$(0.474)(1 - 0.474)(1 - 0.474) = 0.1311$
5	$(0.525)(1 - 0.525)(0.1311)(-0.2) = -0.0065$
4	$(0.332)(1 - 0.332)(0.1311)(-0.3) = -0.0087$

10

## Basic Back-propagation Example (cont. 6): Gradient



x <sub>1</sub>	x <sub>2</sub>	x <sub>3</sub>	W <sub>41</sub>	W <sub>51</sub>	W <sub>42</sub>	W <sub>52</sub>	W <sub>43</sub>	W <sub>53</sub>	W <sub>64</sub>	W <sub>65</sub>	W <sub>40</sub>	W <sub>50</sub>	W <sub>60</sub>
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	-0.4	0.2	0.1

Unit	Activation sum $a$	Unit output $\sigma(a)$
4	$1(0.2) + 0(0.4) + 1(-0.5) + 1(-0.4) = -0.7$	$1/(1+e^{-0.7}) = 0.332$
5	$1(-0.3) + 0(0.1) + 1(0.2) + 1(0.2) = 0.1$	$1/(1+e^{-0.1}) = 0.525$
6	$(-0.3)(0.332) + (-0.2)(0.525) + 1(0.1) = -0.105$	$1/(1+e^{0.105}) = 0.474$

Unit	Error signal $\delta$
6	$(0.474)(1 - 0.474)(1 - 0.474) = 0.1311$
5	$(0.525)(1 - 0.525)(0.1311)(-0.2) = -0.0065$
4	$(0.332)(1 - 0.332)(0.1311)(-0.3) = -0.0087$

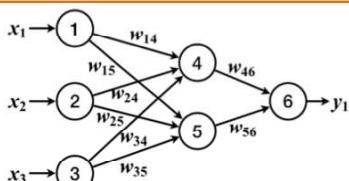
x <sub>1</sub>	x <sub>2</sub>	x <sub>3</sub>
1	0	1

Once all  $\delta$ 's are available,  
can compute gradient

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i \quad \frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j$$

11

## Basic Back-propagation Example (cont. 7): Weights Update



x <sub>1</sub>	x <sub>2</sub>	x <sub>3</sub>	W <sub>41</sub>	W <sub>51</sub>	W <sub>42</sub>	W <sub>52</sub>	W <sub>43</sub>	W <sub>53</sub>	W <sub>64</sub>	W <sub>65</sub>	W <sub>40</sub>	W <sub>50</sub>	W <sub>60</sub>
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	-0.4	0.2	0.1

Unit	Activation sum $a$	Unit output $\sigma(a)$
4	$1(0.2) + 0(0.4) + 1(-0.5) + 1(0.4) = -0.7$	$1/(1+e^{-0.7}) = 0.332$
5	$1(-0.3) + 0(0.1) + 1(0.2) + 1(0.2) = 0.1$	$1/(1+e^{-0.1}) = 0.525$
6	$(-0.3)(0.332) - (0.2)(0.525) + 1(0.1) = -0.105$	$1/(1+e^{0.105}) = 0.474$

Unit	Error signal $\delta$
6	$(0.474)(1 - 0.474)(1 - 0.474) = 0.1311$
5	$(0.525)(1 - 0.525)(0.1311)(-0.2) = -0.0065$
4	$(0.332)(1 - 0.332)(0.1311)(-0.3) = -0.0087$

w( $\tau$ )	Weight update w( $\tau+1$ )
W <sub>64</sub>	$-0.3 + (0.9)(0.1311)(0.332) = -0.261$
W <sub>65</sub>	$-0.2 + (0.9)(0.1311)(0.525) = -0.138$
W <sub>41</sub>	$0.2 + (0.9)(-0.0087)(1) = 0.192$
W <sub>51</sub>	$-0.3 + (0.9)(-0.0065)(1) = -0.306$
W <sub>42</sub>	$0.4 + (0.9)(-0.0087)(0) = 0.4$
W <sub>52</sub>	$0.1 + (0.9)(-0.0065)(0) = 0.1$
W <sub>43</sub>	$-0.5 + (0.9)(-0.0087)(1) = -0.508$
W <sub>53</sub>	$0.2 + (0.9)(-0.0065)(1) = 0.194$
W <sub>64</sub>	$0.1 + (0.9)(0.1311) = 0.218$
W <sub>65</sub>	$0.2 + (0.9)(-0.0065) = 0.194$
W <sub>40</sub>	$-0.4 + (0.9)(-0.0087) = -0.408$

$$w(\tau + 1) = w(\tau) - \eta \nabla_w E$$

Gradient components computed as  $\delta_j x_i$  and  $\delta_k z_j$

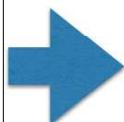
Plus-sign in weight-update because of earlier use of  $(t_k - y_k)$  instead of  $(y_k - t_k)$

12

- Up to now:
  - Covered foundations of basic back-propagation
- In upcoming lectures:
  - Will cover modern back-propagation techniques, exploiting computational graphs, currently used in training of deep neural networks

## This Lecture

---



- Basic back-propagation example
- Jacobian and Hessian

# Jacobian Matrix

---

- Given function

$$f : \mathbb{R}^m \rightarrow \mathbb{R}^n$$

- Jacobian matrix is defined as

$$J \in \mathbb{R}^{n \times m}$$

$$J_{i,j} = \frac{\partial}{\partial x_j} f(\mathbf{x})_i$$

# Second Derivative (Recap)

---

- Given function  $f : \mathbb{R}^n \rightarrow \mathbb{R}$

- Derivative w.r.t.  $x_i$  of derivative of  $f$  w.r.t.  $x_j$

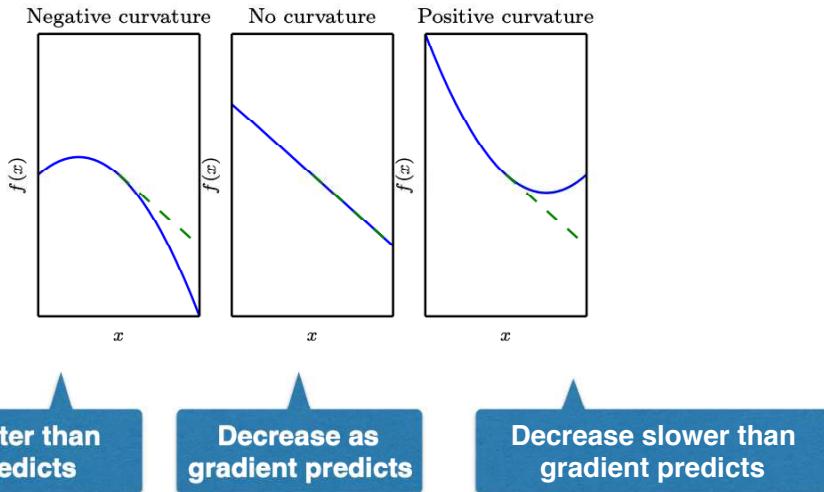
$$\frac{\partial^2}{\partial x_i \partial x_j} f$$

- In single dimension simply  $f''(x)$

$$\frac{d^2}{dx^2} f$$

# Second Derivative and Curvature

- Example: quadratic functions of different curvatures
  - Dashed line — cost function value to be expected on step downhill when based only on gradient information



# Hessian

- First derivative of  $E$  w.r.t. vector
  - Gradient
  - Example: Consider  $w$  for two-dimensional  $w = [w_1, w_2]^\top$
- Second derivative
  - Hessian
- Hessian for two-dimensional space
  - We will see  $H$  for n-dimensional space shortly
  - Note that Hessian is Jacobian of gradient

$$\nabla E(w) = \frac{d}{dw} E(w) = \begin{bmatrix} \frac{\partial E}{\partial w_1} \\ \frac{\partial E}{\partial w_2} \end{bmatrix}$$

$$H = \nabla \nabla E(w) = \begin{bmatrix} \frac{\partial^2 E}{\partial w_1^2} & \frac{\partial^2 E}{\partial w_1 \partial w_2} \\ \frac{\partial^2 E}{\partial w_2 \partial w_1} & \frac{\partial^2 E}{\partial w_2^2} \end{bmatrix}$$

# Questions?

---

---

# **Neural Networks and Deep Learning**

**Dr. Jerome J. Braun**

## **This Lecture: Computational Graphs II**

Course: Neural Networks and Deep Learning  
IE 7615

---

---

**Unauthorized distributing, redistributing, posting,  
and/or reposting, of the materials of this course  
(including course lectures, lecture slides, slide-sets,  
syllabi, guidelines, assignments, quizzes, exams,  
presentations, software, electronic media, etc.),  
is prohibited.**

# Recap: Computational Graph

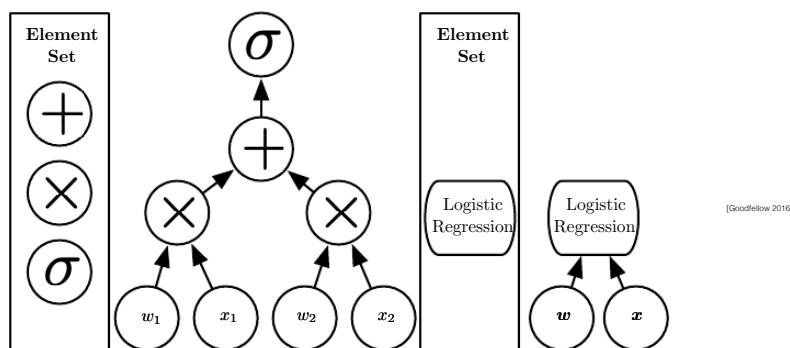
- **Graph  $G(V,E)$** 
  - $V$  — set of vertices (nodes)
  - $E$  — set of edges (connections between vertices)
- **Directed vs. undirected graphs**
- **Computational graph**
  - Directed graph
  - Nodes may represent
    - Operations
    - Structures, e.g., variables
    - Structures *and* operations

J. Braun

3

# Recap: Computational Graphs

- Consider computational graph with nodes representing mathematical operations
  - Nodes may be at different levels of “complexity”
  - Example:



[Goodfellow 2016]

J. Braun

4

# Math Operations and Computational Graphs

---

- **Nodes representing mathematical operations**

- Examples of simple operations and corresponding derivatives

$$f(x, y) = x + y \quad \frac{\partial f}{\partial x} = 1 \quad \frac{\partial f}{\partial y} = 1$$

$$f(x, y) = xy \quad \frac{\partial f}{\partial x} = y \quad \frac{\partial f}{\partial y} = x$$

$$f(x, y) = \max(x, y) \quad \frac{\partial f}{\partial x} = 1 \quad (x \geq y) \quad \frac{\partial f}{\partial y} = 1 \quad (y \geq x)$$

- **Can decompose functions into simpler components, e.g.,**

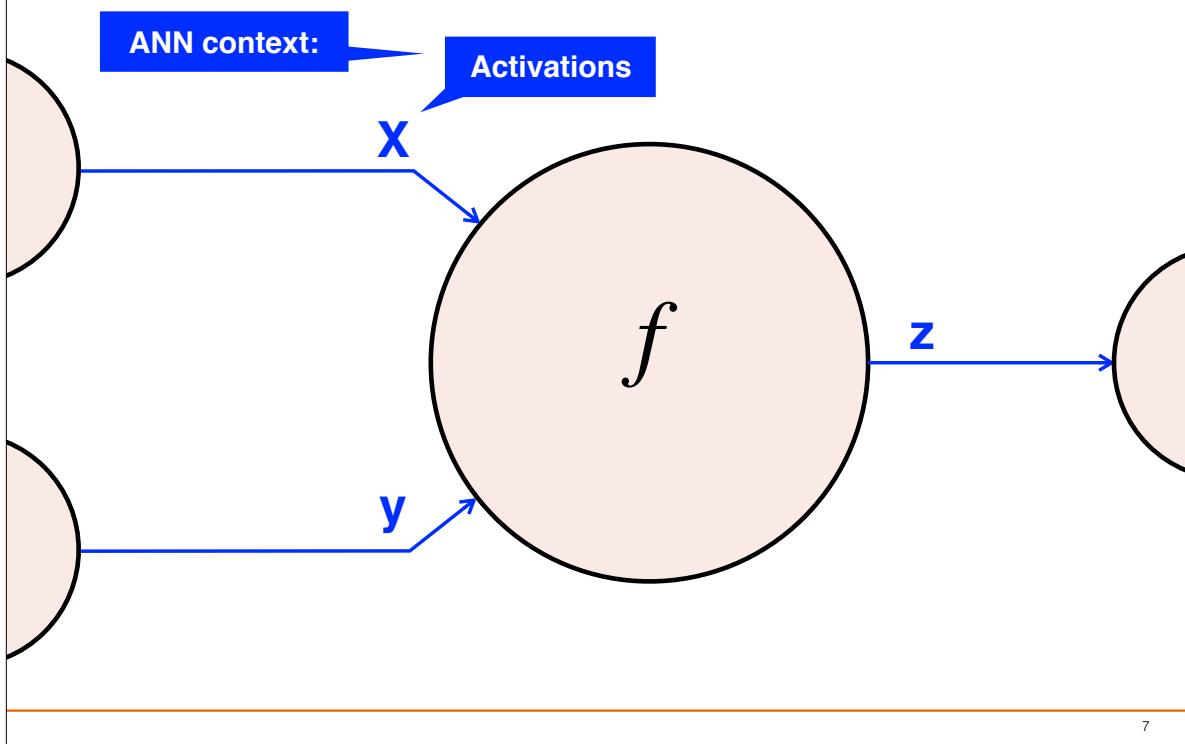
$$f(x, y) = (x + y)z \quad a = x + y \quad f = az$$

# Flows through Computational Graph

---

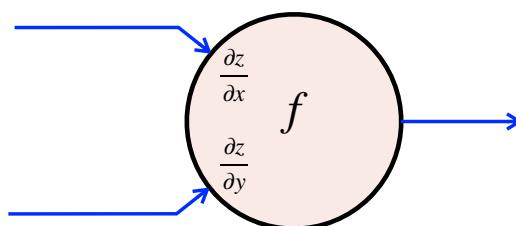
- **Executing procedures defined by computational graph**
- **Forward pass**
  - Inputs to graph propagated from inputs to output via directed edges
- **Backward (reverse) pass**
  - From output to inputs
  - To obtain derivatives (gradient)

# Forward Pass



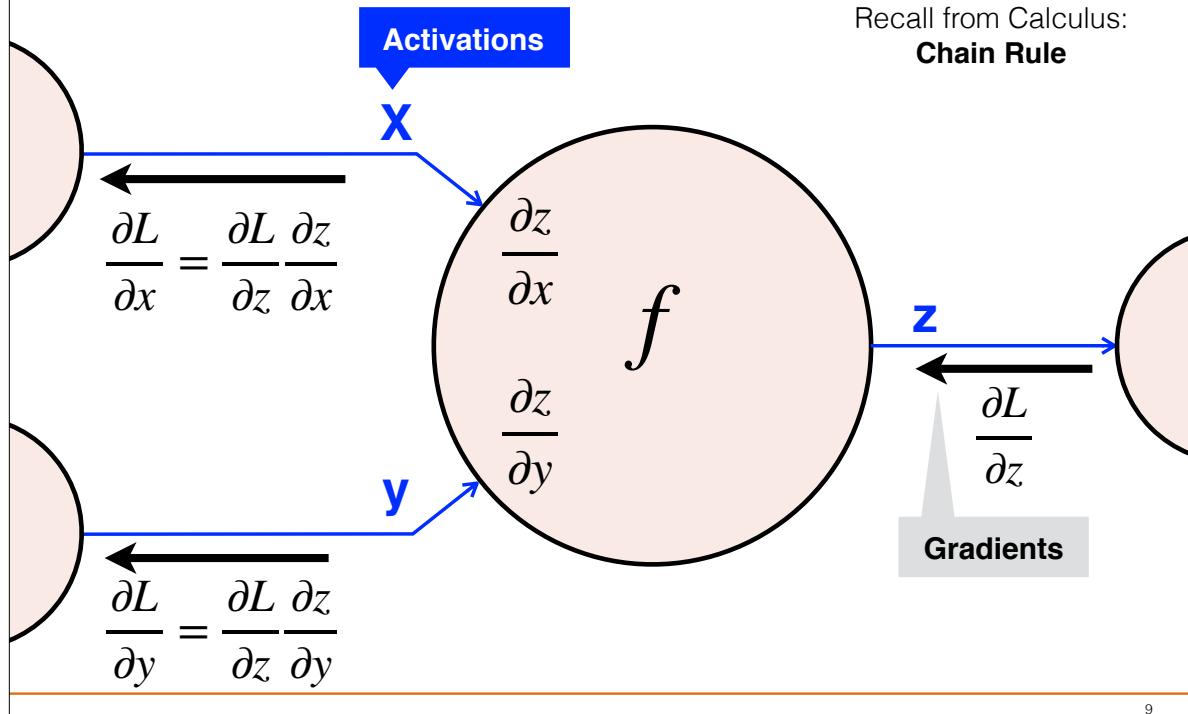
7

# Gradients at Graph Nodes



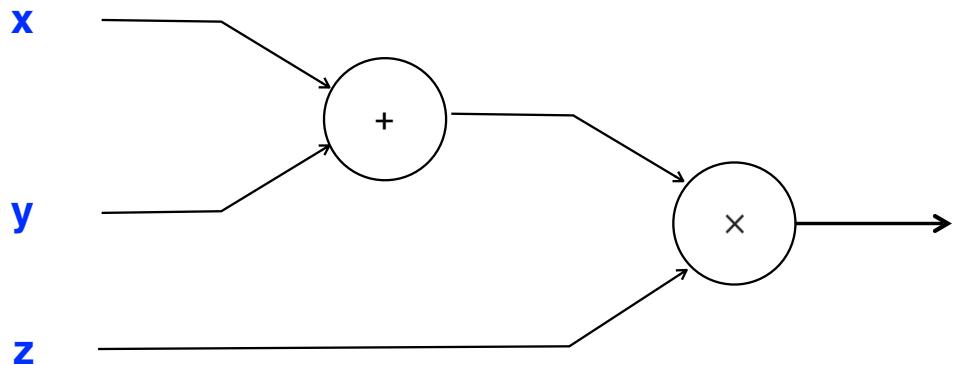
- “Local” at graph nodes
- Examples
  - ADD
    - Gradients on outputs distributed equally on inputs
      - Because  $\text{partial\_Derivative}(x+y+\dots) = 1$
  - MULT
    - Gradients on inputs equal to “switched input” multiplied by gradient on output
  - MAX
    - Gradient distributed to input which had highest value on forward pass

# Back-Propagating Gradients



9

## Back-propagating Gradients in Computational Graph – Example



10

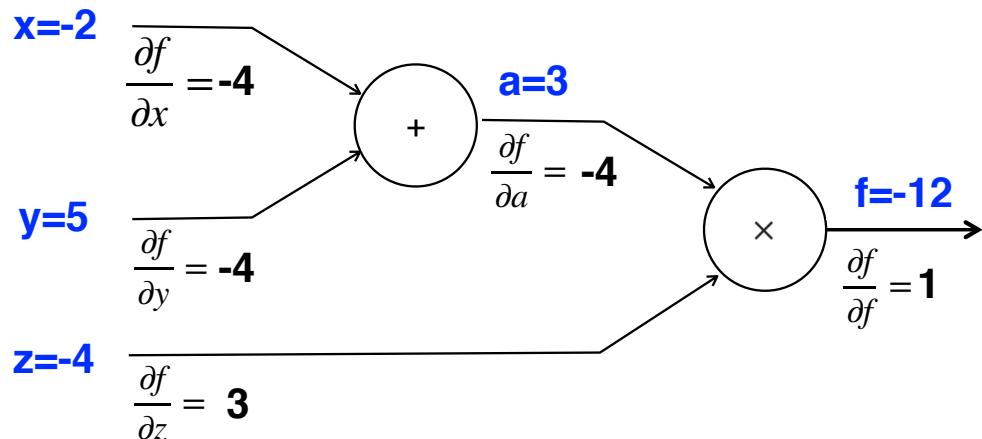
## Back-propagating Gradients in Computational Graph – Example

$$f = (x+y)z \quad a = x+y$$

$$\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$$

$$f = az \quad \frac{\partial f}{\partial a} = z \quad \frac{\partial f}{\partial z} = a$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial a} \frac{\partial a}{\partial x} \quad \frac{\partial f}{\partial y} = \frac{\partial f}{\partial a} \frac{\partial a}{\partial y} \quad \frac{\partial f}{\partial z} = \frac{\partial f}{\partial a} \frac{\partial a}{\partial z}$$



11

Questions?

---

# **Neural Networks and Deep Learning**

**Dr. Jerome J. Braun**

## **This Lecture: Computational Graphs III**

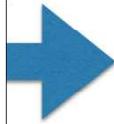
Course: Neural Networks and Deep Learning  
IE 7615

---

---

**Unauthorized distributing, redistributing, posting,  
and/or reposting, of the materials of this course  
(including course lectures, lecture slides, slide-sets,  
syllabi, guidelines, assignments, quizzes, exams,  
presentations, software, electronic media, etc.),  
is prohibited.**

# This Lecture



- Computational graph example
- Flow through computational graph example
  - Forward pass
  - Backward (reverse) pass
    - Gradients

J. Braun

3

## Recap: Computational-Graph Example

$$f = (x+y)z \quad a = x+y$$

$$\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$$

$$f = az \quad \frac{\partial f}{\partial a} = z \quad \frac{\partial f}{\partial z} = a$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial a} \frac{\partial a}{\partial x}$$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial a} \frac{\partial a}{\partial y}$$

$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial a} \frac{\partial a}{\partial z}$$

$$x = -2$$

$$\frac{\partial f}{\partial x} = -4$$

$$a = 3$$

$$y = 5$$

$$\frac{\partial f}{\partial y} = -4$$

$$f = -12$$

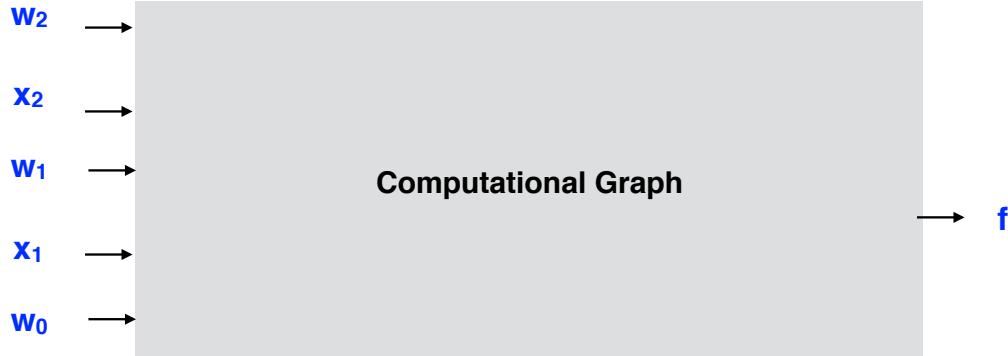
$$z = -4$$

$$\frac{\partial f}{\partial z} = 3$$

$$\frac{\partial f}{\partial f} = 1$$

4

**Computational Graph for**  $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$

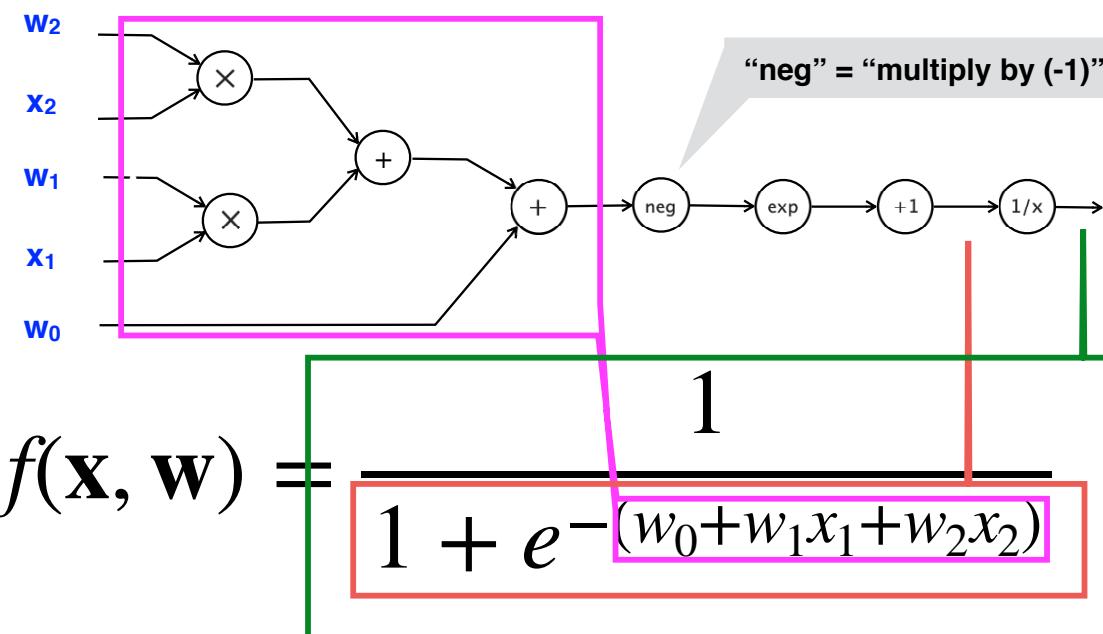


$$f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$$

- Want gradients
- Values of partial derivatives of  $f$

5

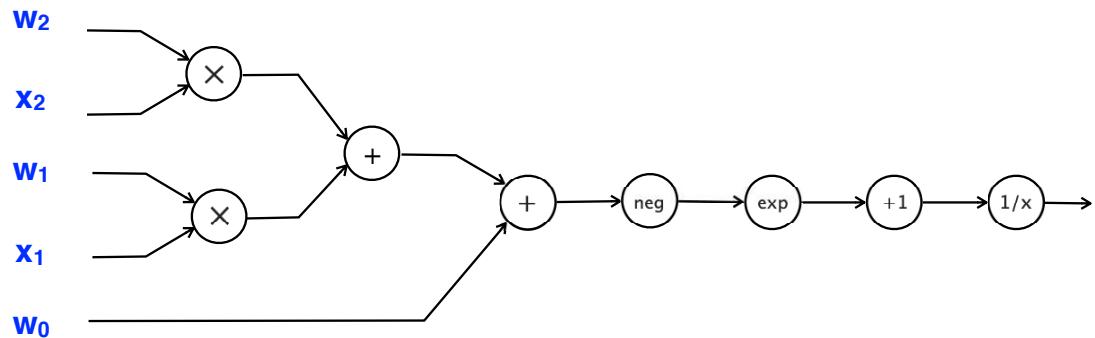
**Computational Graph for**  $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$



6

**Computational Graph for**  $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$

---

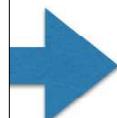


7

## This Lecture

---

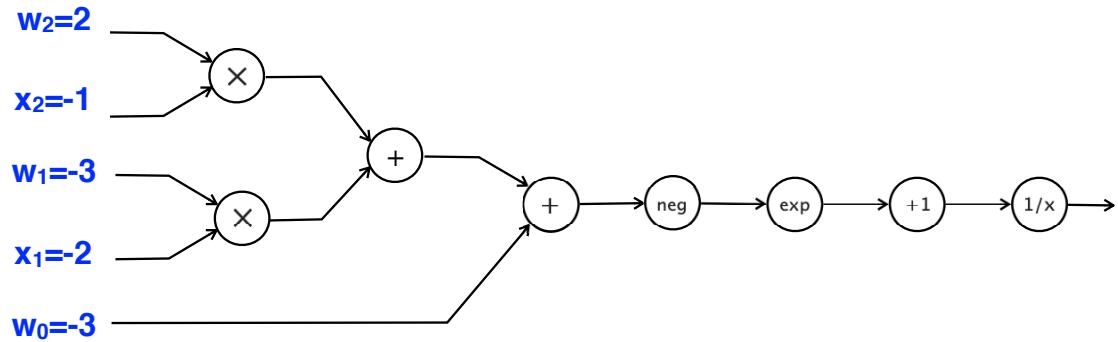
---



- Computational graph example
- Flow through computational graph example
  - Forward pass
  - Backward (reverse) pass
    - Gradients

**Computational Graph for**  $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$

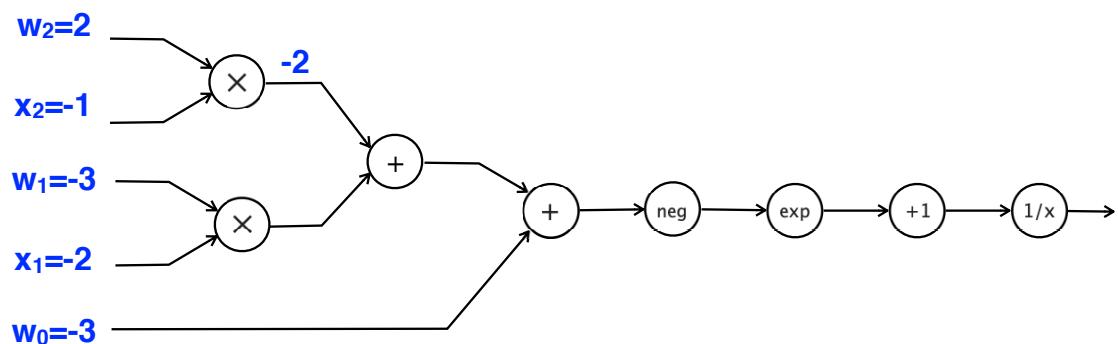
---



9

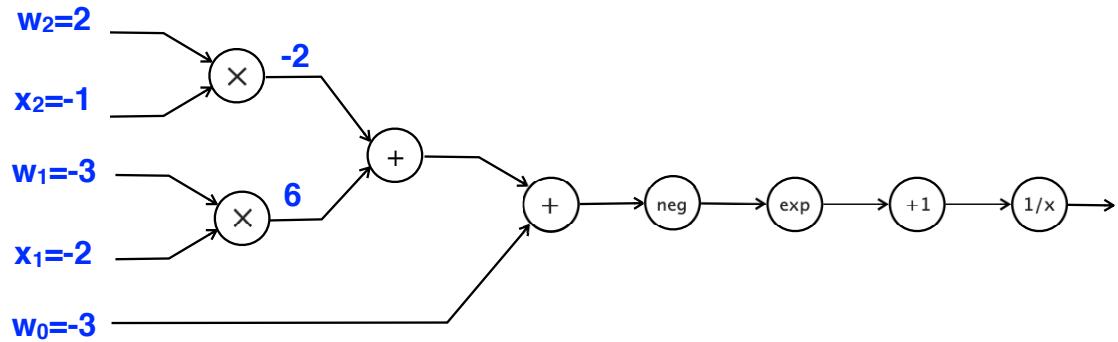
**Computational Graph for**  $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$

---



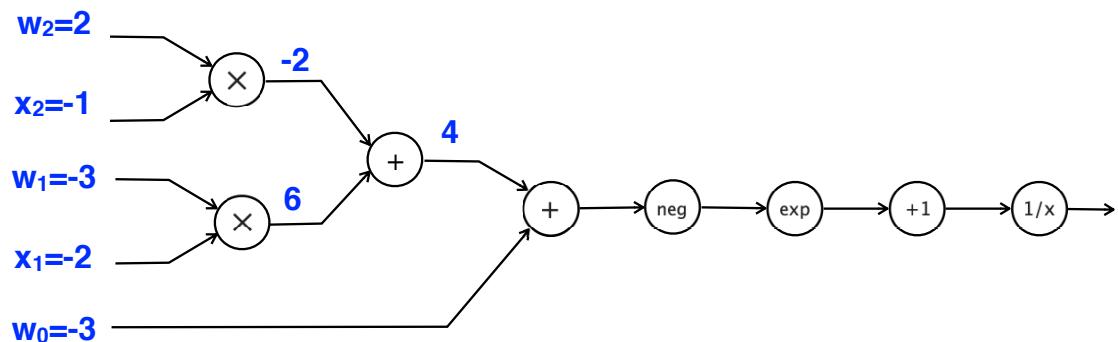
10

**Computational Graph for**  $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$



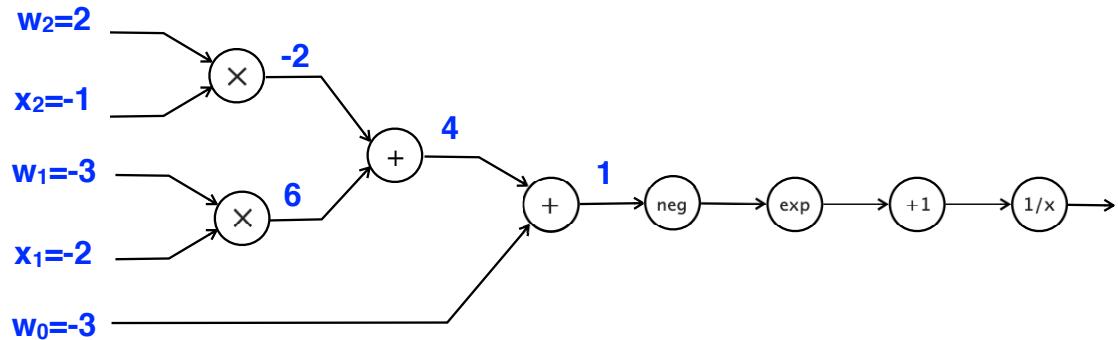
11

**Computational Graph for**  $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$



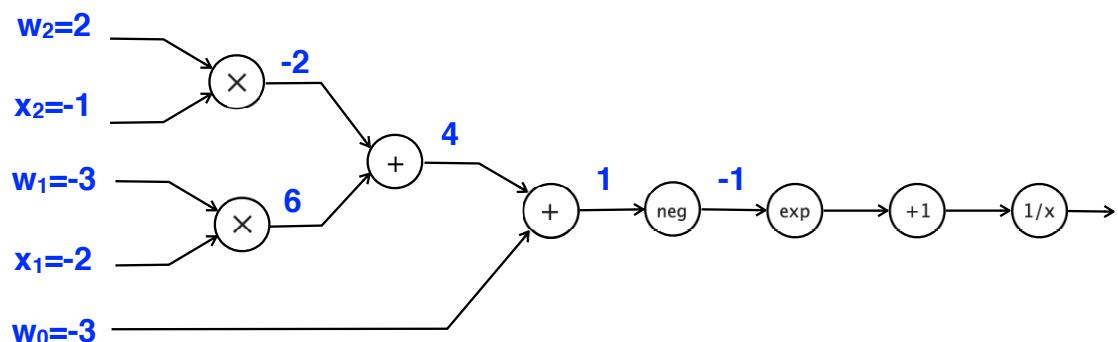
12

**Computational Graph for**  $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$



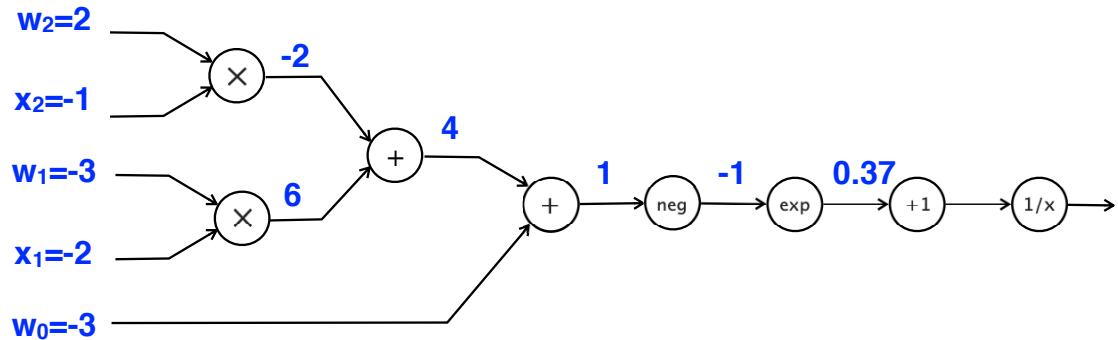
13

**Computational Graph for**  $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$



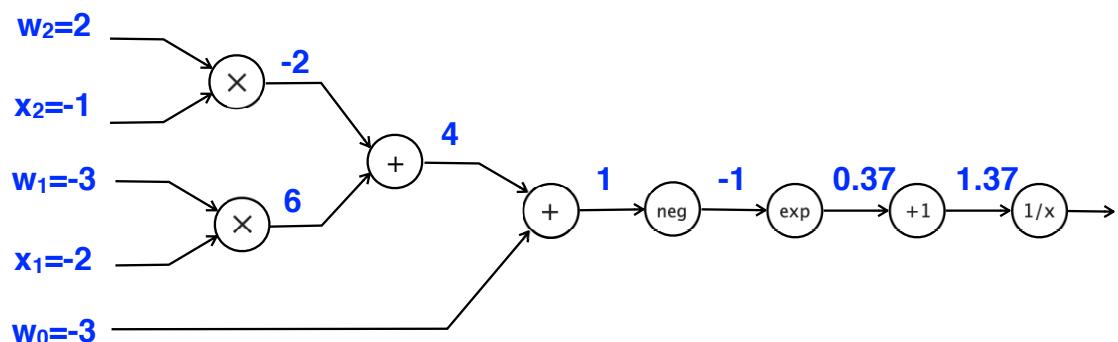
14

**Computational Graph for**  $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$



15

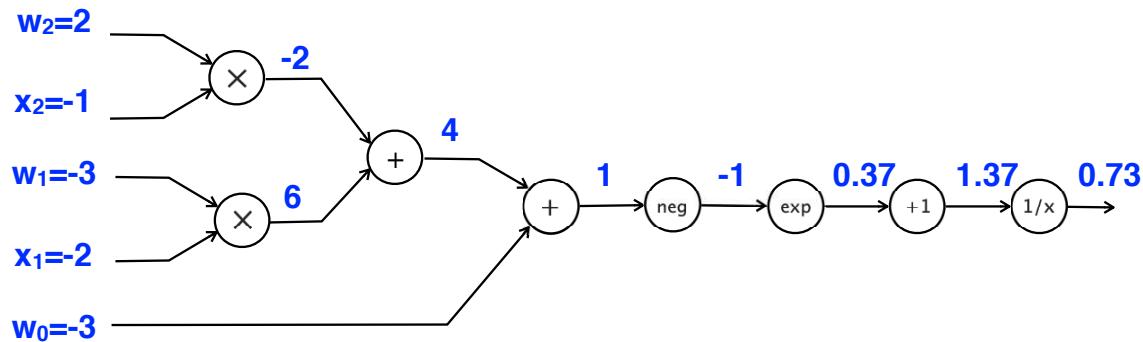
**Computational Graph for**  $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$



16

**Computational Graph for**  $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$

---

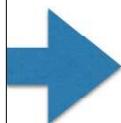


17

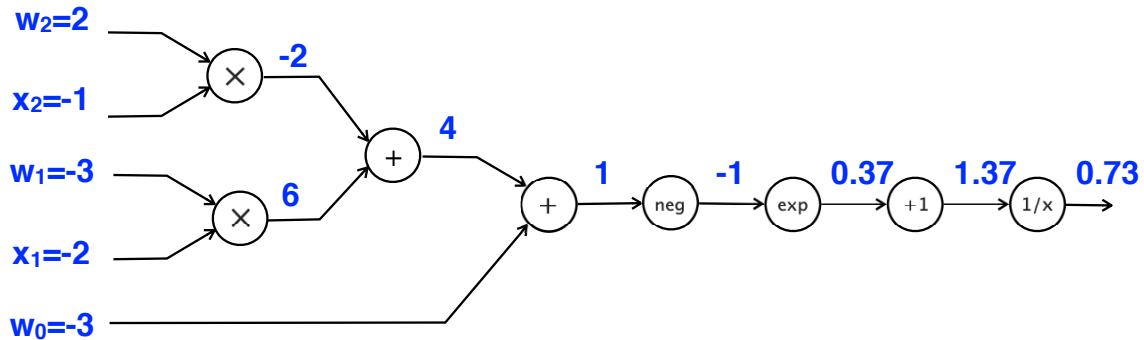
## This Lecture

---

- Computational graph example
- Flow through computational graph example
  - Forward pass
  - Backward (reverse) pass
  - Gradients



## Computational Graph for $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$



$$f(x) = e^x \quad \frac{df}{dx} = e^x$$

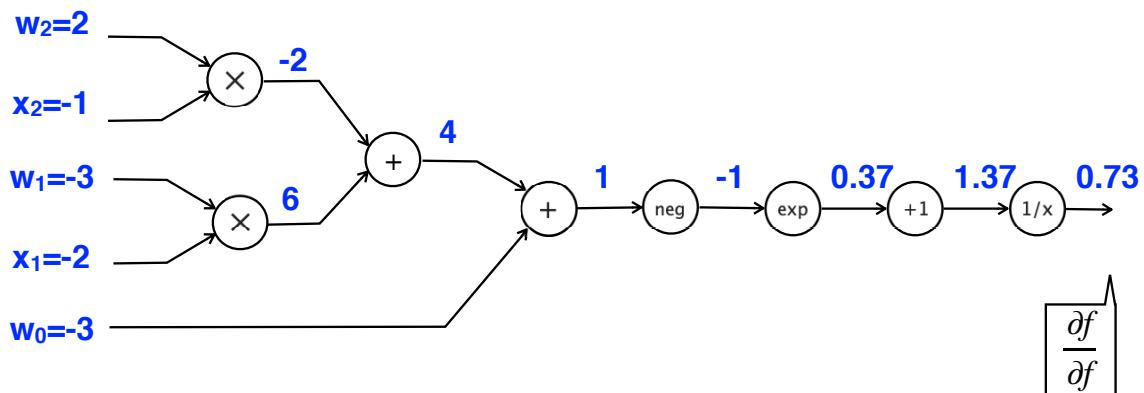
$$f(x) = \frac{1}{x} \quad \frac{df}{dx} = -\frac{1}{x^2}$$

$$f_c(x) = cx \quad \frac{df_c}{dx} = c$$

$$f_c(x) = x + c \quad \frac{df_c}{dx} = 1$$

19

## Computational Graph for $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$



$$f(x) = e^x \quad \frac{df}{dx} = e^x$$

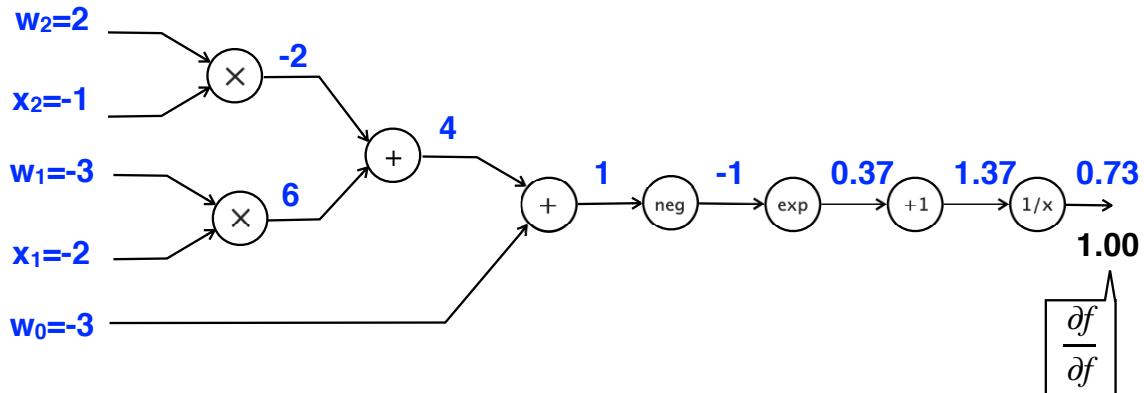
$$f(x) = \frac{1}{x} \quad \frac{df}{dx} = -\frac{1}{x^2}$$

$$f_c(x) = cx \quad \frac{df_c}{dx} = c$$

$$f_c(x) = x + c \quad \frac{df_c}{dx} = 1$$

20

## Computational Graph for $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$



$$f(x) = e^x \quad \frac{df}{dx} = e^x$$

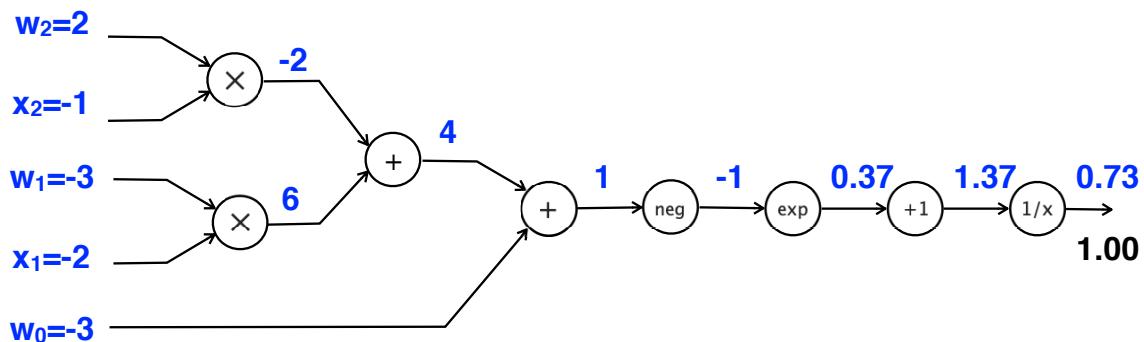
$$f(x) = \frac{1}{x} \quad \frac{df}{dx} = -\frac{1}{x^2}$$

$$f_c(x) = cx \quad \frac{df_c}{dx} = c$$

$$f_c(x) = x + c \quad \frac{df_c}{dx} = 1$$

21

## Computational Graph for $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$



$$f(x) = e^x \quad \frac{df}{dx} = e^x$$

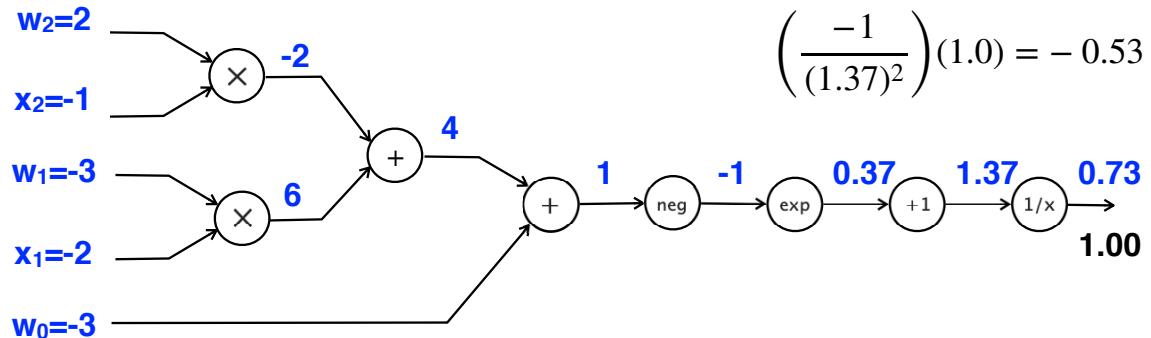
$$f(x) = \frac{1}{x} \quad \frac{df}{dx} = -\frac{1}{x^2}$$

$$f_c(x) = cx \quad \frac{df_c}{dx} = c$$

$$f_c(x) = x + c \quad \frac{df_c}{dx} = 1$$

22

## Computational Graph for $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$



$$f(x) = e^x \quad \frac{df}{dx} = e^x$$

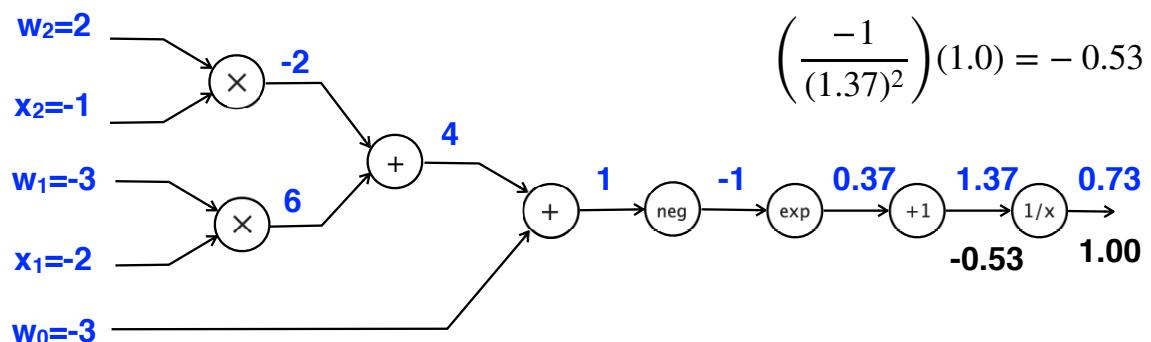
$$f(x) = \frac{1}{x} \quad \frac{df}{dx} = -\frac{1}{x^2}$$

$$f_c(x) = cx \quad \frac{df_c}{dx} = c$$

$$f_c(x) = x + c \quad \frac{df_c}{dx} = 1$$

23

## Computational Graph for $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$



$$f(x) = e^x \quad \frac{df}{dx} = e^x$$

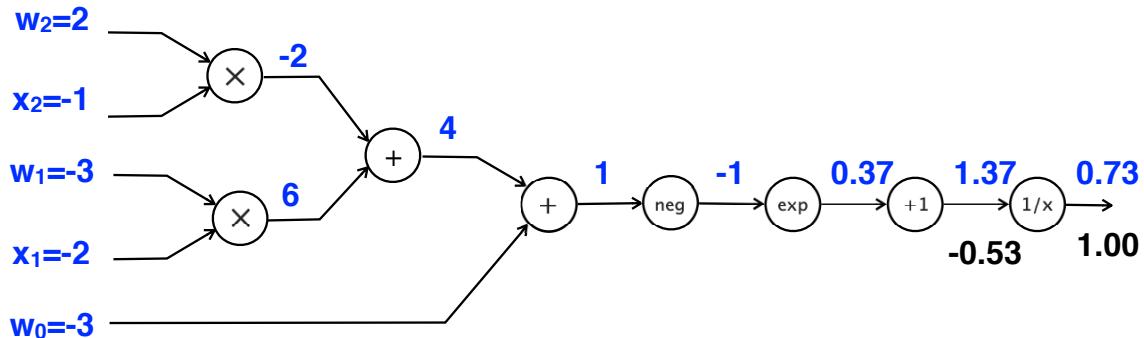
$$f(x) = \frac{1}{x} \quad \frac{df}{dx} = -\frac{1}{x^2}$$

$$f_c(x) = cx \quad \frac{df_c}{dx} = c$$

$$f_c(x) = x + c \quad \frac{df_c}{dx} = 1$$

24

## Computational Graph for $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$



$$f(x) = e^x \quad \frac{df}{dx} = e^x$$

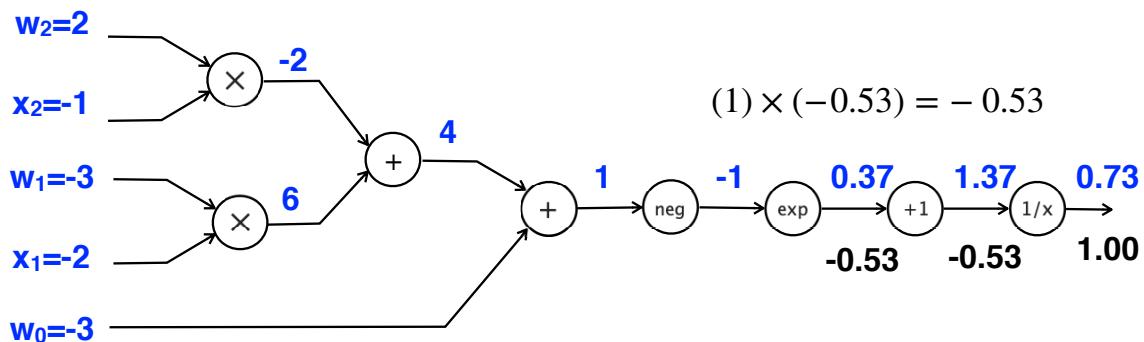
$$f(x) = \frac{1}{x} \quad \frac{df}{dx} = -\frac{1}{x^2}$$

$$f_c(x) = cx \quad \frac{df_c}{dx} = c$$

$$f_c(x) = x + c \quad \frac{df_c}{dx} = 1$$

25

## Computational Graph for $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$



$$f(x) = e^x \quad \frac{df}{dx} = e^x$$

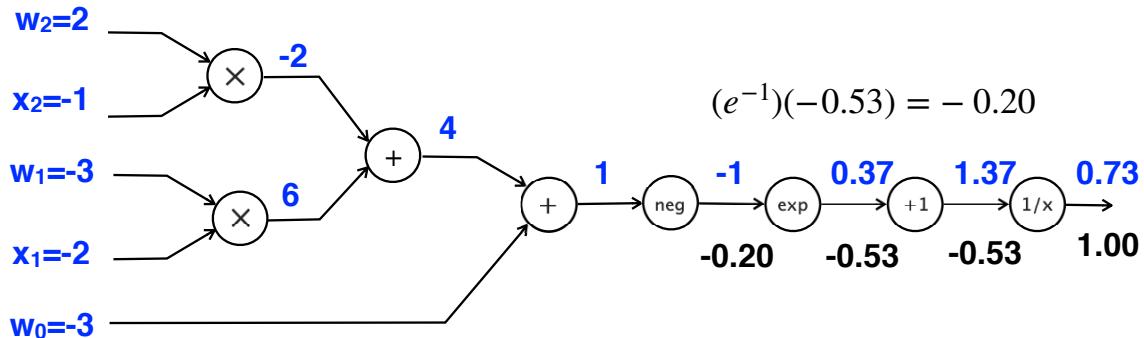
$$f(x) = \frac{1}{x} \quad \frac{df}{dx} = -\frac{1}{x^2}$$

$$f_c(x) = cx \quad \frac{df_c}{dx} = c$$

$$f_c(x) = x + c \quad \frac{df_c}{dx} = 1$$

26

## Computational Graph for $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$



$$f(x) = e^x \quad \frac{df}{dx} = e^x$$

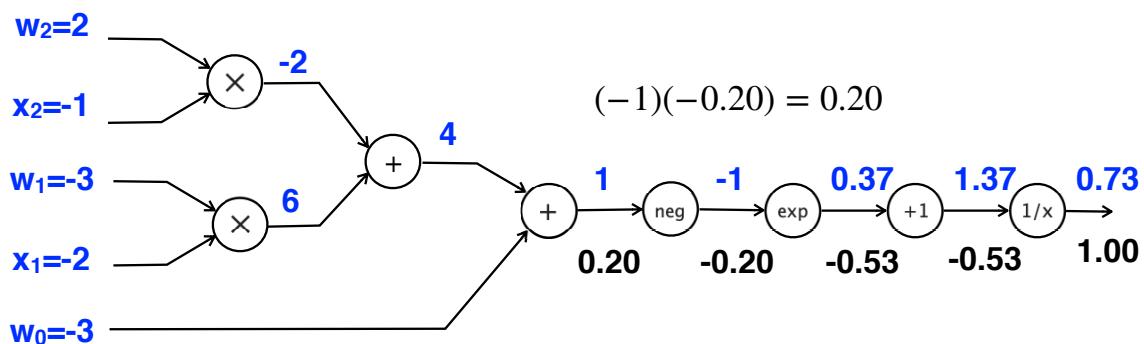
$$f(x) = \frac{1}{x} \quad \frac{df}{dx} = -\frac{1}{x^2}$$

$$f_c(x) = cx \quad \frac{df_c}{dx} = c$$

$$f_c(x) = x + c \quad \frac{df_c}{dx} = 1$$

27

## Computational Graph for $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$



$$f(x) = e^x \quad \frac{df}{dx} = e^x$$

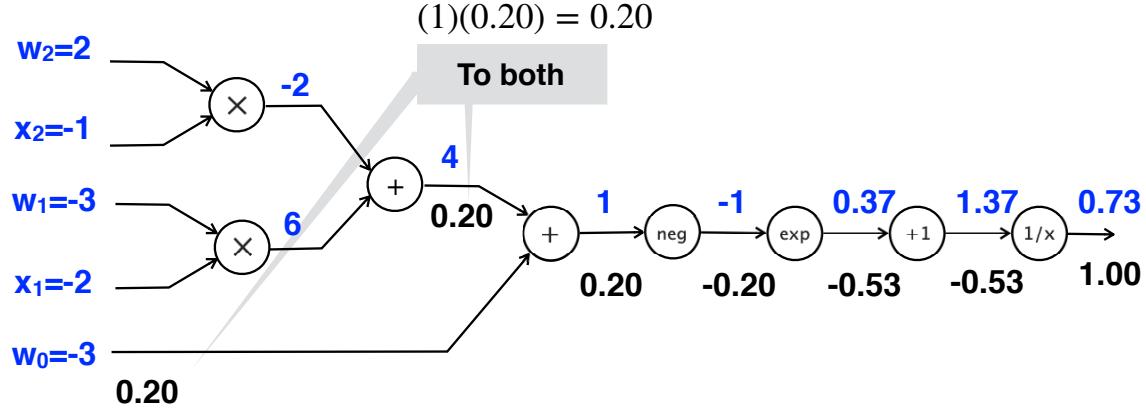
$$f(x) = \frac{1}{x} \quad \frac{df}{dx} = -\frac{1}{x^2}$$

$$f_c(x) = cx \quad \frac{df_c}{dx} = c$$

$$f_c(x) = x + c \quad \frac{df_c}{dx} = 1$$

28

## Computational Graph for $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$



$$f(x) = e^x \quad \frac{df}{dx} = e^x$$

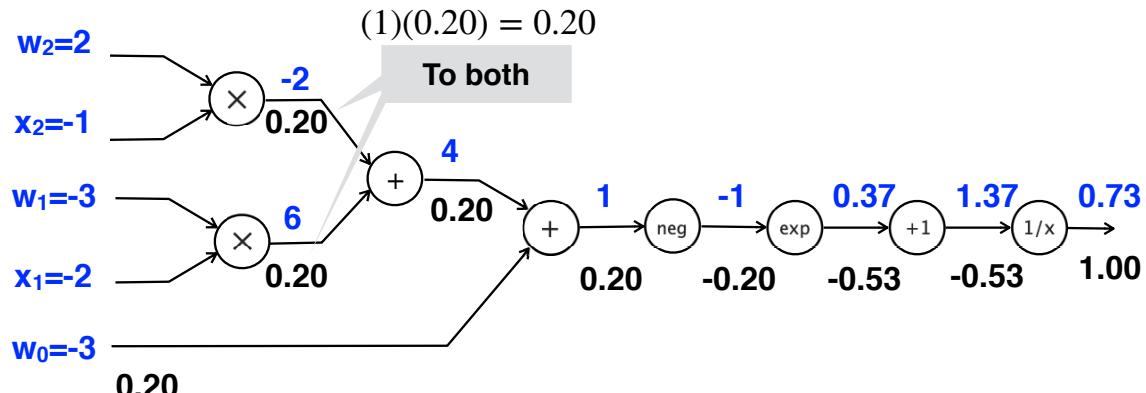
$$f(x) = \frac{1}{x} \quad \frac{df}{dx} = -\frac{1}{x^2}$$

$$f_c(x) = cx \quad \frac{df_c}{dx} = c$$

$$f_c(x) = x + c \quad \frac{df_c}{dx} = 1$$

29

## Computational Graph for $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$



$$f(x) = e^x \quad \frac{df}{dx} = e^x$$

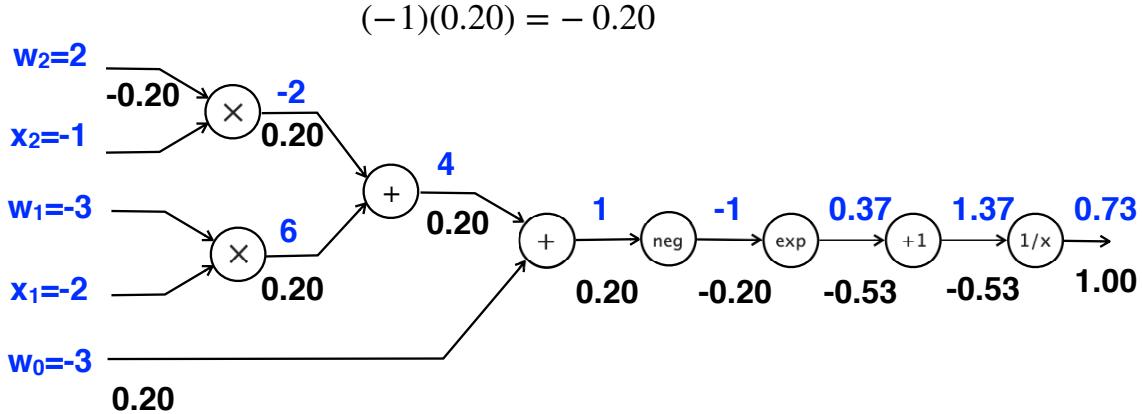
$$f(x) = \frac{1}{x} \quad \frac{df}{dx} = -\frac{1}{x^2}$$

$$f_c(x) = cx \quad \frac{df_c}{dx} = c$$

$$f_c(x) = x + c \quad \frac{df_c}{dx} = 1$$

30

## Computational Graph for $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$



$$f(x) = e^x \quad \frac{df}{dx} = e^x$$

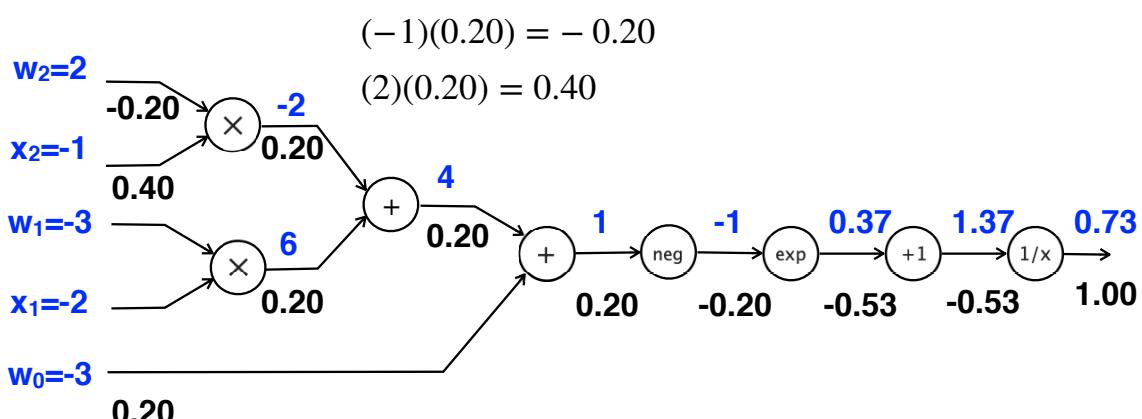
$$f(x) = \frac{1}{x} \quad \frac{df}{dx} = -\frac{1}{x^2}$$

$$f_c(x) = cx \quad \frac{df_c}{dx} = c$$

$$f_c(x) = x + c \quad \frac{df_c}{dx} = 1$$

31

## Computational Graph for $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$



$$f(x) = e^x \quad \frac{df}{dx} = e^x$$

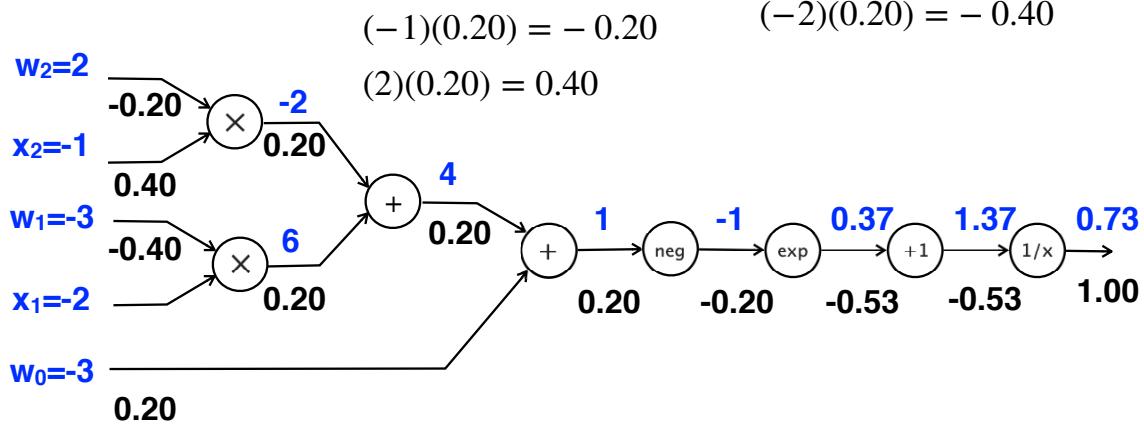
$$f(x) = \frac{1}{x} \quad \frac{df}{dx} = -\frac{1}{x^2}$$

$$f_c(x) = cx \quad \frac{df_c}{dx} = c$$

$$f_c(x) = x + c \quad \frac{df_c}{dx} = 1$$

32

## Computational Graph for $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$



$$f(x) = e^x \quad \frac{df}{dx} = e^x$$

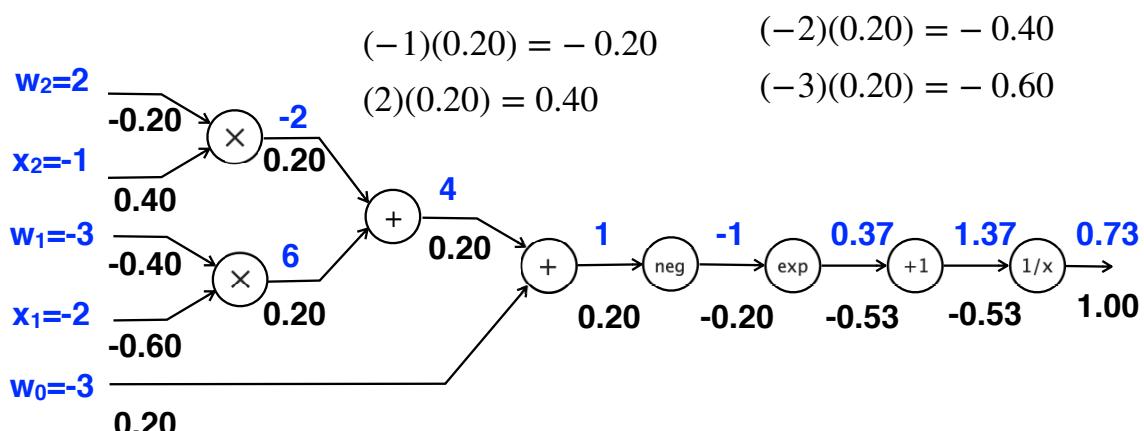
$$f(x) = \frac{1}{x} \quad \frac{df}{dx} = -\frac{1}{x^2}$$

$$f_c(x) = cx \quad \frac{df_c}{dx} = c$$

$$f_c(x) = x + c \quad \frac{df_c}{dx} = 1$$

33

## Computational Graph for $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$



$$f(x) = e^x \quad \frac{df}{dx} = e^x$$

$$f(x) = \frac{1}{x} \quad \frac{df}{dx} = -\frac{1}{x^2}$$

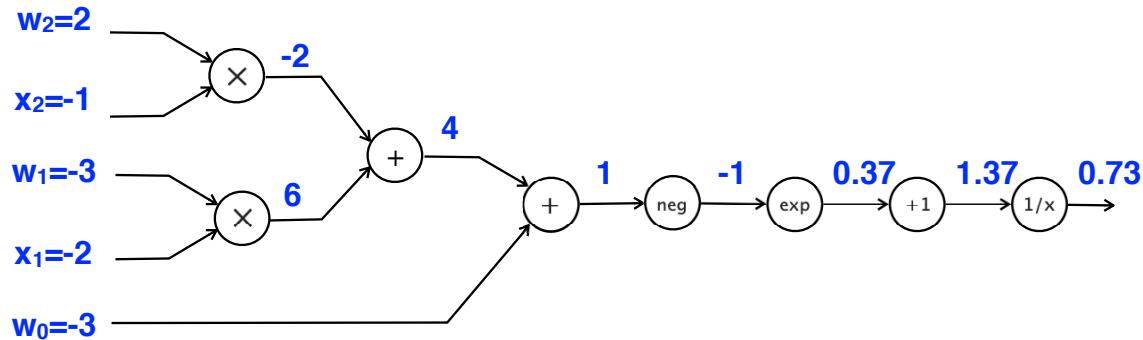
$$f_c(x) = cx \quad \frac{df_c}{dx} = c$$

$$f_c(x) = x + c \quad \frac{df_c}{dx} = 1$$

34

**Computational Graph for**  $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$

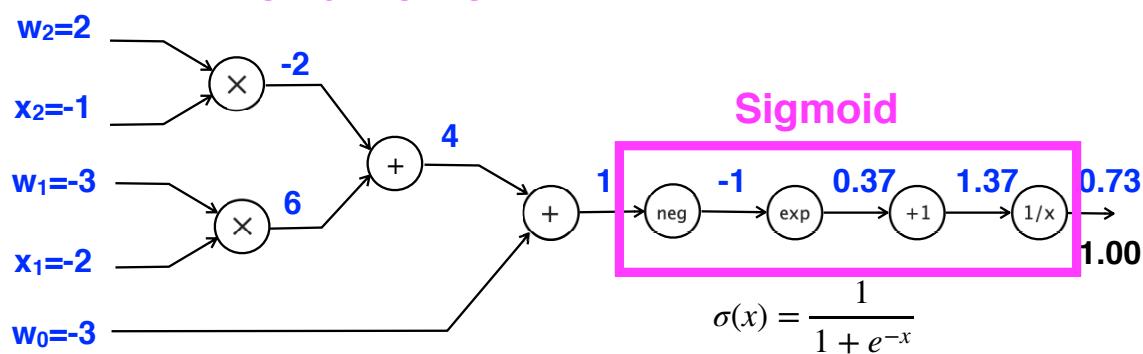
**Alternative view:**



35

**Computational Graph for**  $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$

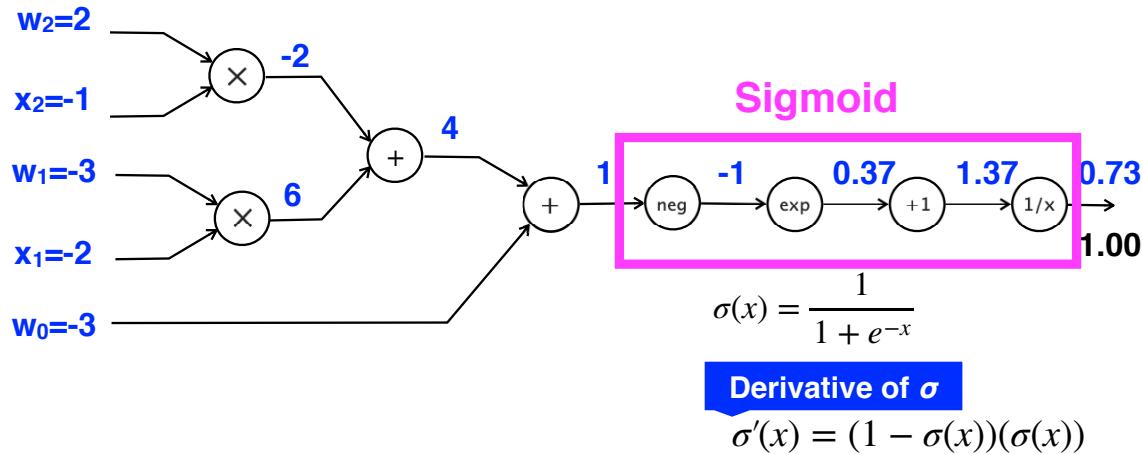
**Alternative view:**



36

**Computational Graph for**  $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$

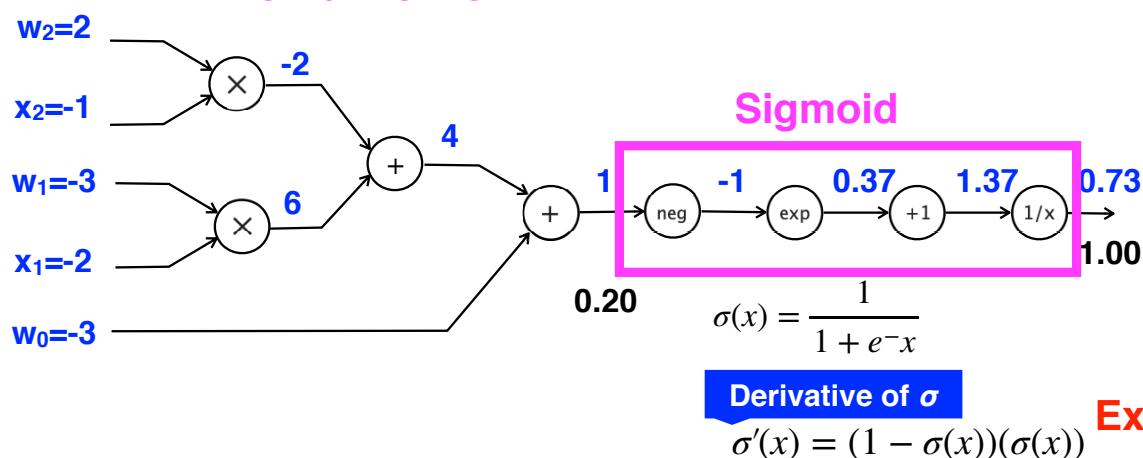
**Alternative view:**



37

**Computational Graph for**  $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$

**Alternative view:**

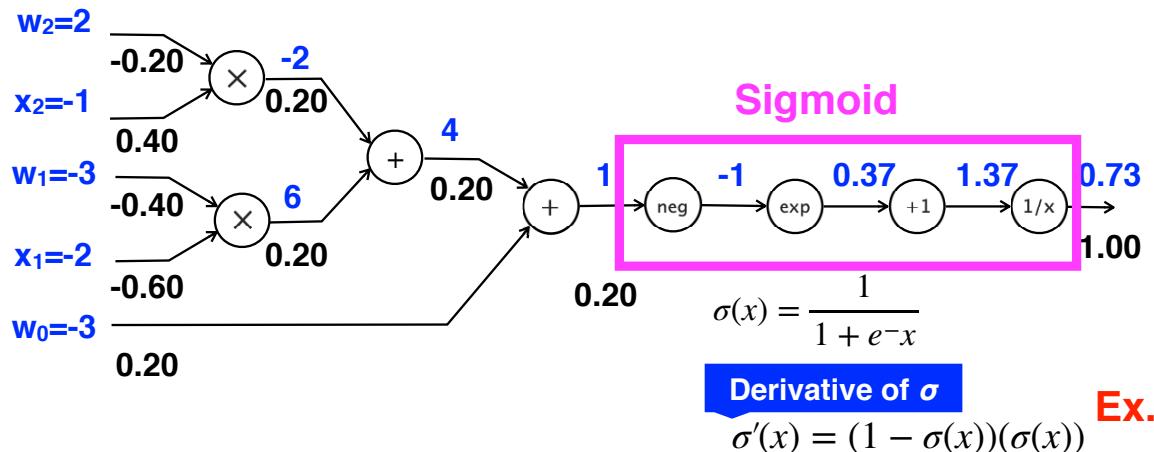


**Ex.**

38

**Computational Graph for**  $f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}$

**Alternative view:**



**Derivative of  $\sigma$**

$$\sigma'(x) = (1 - \sigma(x))(\sigma(x))$$

**Ex.**

$$(1 - 0.73)(0.73) = 0.2$$

39

**Questions?**

---

# **Neural Networks and Deep Learning**

**Dr. Jerome J. Braun**

## **This Lecture: TensorFlow Essentials I**

Course: Neural Networks and Deep Learning  
IE 7615

---

J. Braun

---

**Unauthorized distributing, redistributing, posting,  
and/or reposting, of the materials of this course  
(including course lectures, lecture slides, slide-sets,  
syllabi, guidelines, assignments, quizzes, exams,  
presentations, software, electronic media, etc.),  
is prohibited.**

# TensorFlow Basics

---

- TensorFlow — until recently, most-popular deep-learning library
- TensorFlow maintains its leading position in industry
  - In academia, position of TensorFlow currently challenged by PyTorch
- “Look and feel” similar to NumPy
- Very rich set of building blocks (functions) for implementing
  - Deep networks
  - Variety of other Machine Learning constructs
  - Also applicable for other computationally-intensive tasks
- Support of
  - GPUs (and TPUs)
  - Distributed processing — devices, servers

## TensorFlow Basics (cont.)

---

- Computation graphs
  - JIT-compiler-like construction of computation graphs
  - Focus on optimizing run-time performance
    - Includes exploiting parallelization
  - Portable format of computation graphs
    - Allows training and subsequent operation in different environments
      - E.g.,
        - Train on Linux+GPU platform
        - Then operate under Android+Java

# Installing TensorFlow (TF)

## Assumed:

- You have installed python
- You set up virtual environment as recommended earlier in this course
  - At least basic one such as venv
- Currently in virtual environment
- Numpy and matplotlib installed prior to installing TF

```
pip install --upgrade pip
```

Make sure pip is up to date

```
pip install tensorflow
```

Current STABLE version

```
pip install tf-nightly
```

Potentially UNSTABLE.  
NOT for use by non-experts

## More specifically (Linux and macOS):

```
python3 -m pip install --upgrade tensorflow
```

```
python3.x -m pip install --upgrade tensorflow
```

If you have multiple versions of python: 3.x is version number (e.g., 3.9)

5

# TensorFlow Basics (cont.)

- **tf.Variable** — variables
  - Values mutable
- **tf.constant** — constants
  - Type: `tf.Tensor`
  - Values immutable
  - Can also store strings
- Other TF data structures and operations on them, e.g.,
  - `tf.SparseTensor` — sparse tensors (mostly 0) and operations on sparse tensors
  - `tf.TensorArray` — tensor arrays
  - `tf.ragged.constant` — ragged dimensions (different sizes of slices along them)
  - ...
- **Typing and type conversions**
  - In TF — explicit (manual) type conversions
    - Else exception raised

## TENSORS in TF:

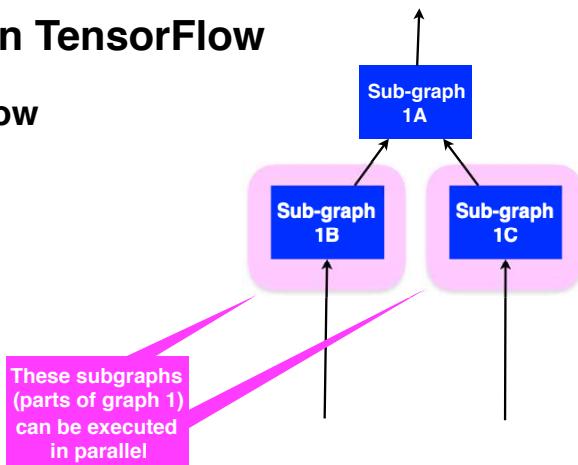
- Tensor objects are fundamental data structures
- Typed multidimensional arrays

## Precision:

- NumPy default: 64 bits
- TF default: 32 bits
- Usually sufficient in ANNs
- More efficient

# TensorFlow Basics (cont.)

- Recall computational graphs
  - Introduced in previous lectures
- Computation graphs in TensorFlow
  - Key feature in TensorFlow



# TensorFlow Evolution

- In TensorFlow 1.x
  - Default: Static graph
  - Optional: Dynamic graph (eager mode)
- Computation graph constructed prior to its evaluation
  - Code “looks” like typical Python programs, but computation not actually executed yet
    - + Computation graph is created
    - + But nothing is computed yet
      - Variables not initialized yet
    - + Evaluation deferred to “session”
- Sessions
  - Perform actual computations defined by computation graph
    - + Variables initialized
    - + Evaluation of computation-graph

You should use Release 2  
You should use most-recent version  
that is available as STABLE release  
(release 2.11.1 or later stable version)

Example:

```
import tensorflow as tf
x = tf.Variable(5, name="x")
y = tf.Variable(7, name="y")
z = x*x + y*y
```

Example:

```
with tf as sess:
    x.initializer.run()
    ...
    result = z.eval()
```

TensorFlow 1.x is now “legacy” and obsolete.

Use TensorFlow 2.x.

# Introduction To TensorFlow 2.x Basics

- **TensorFlow 2.x**

- Default: Dynamic graph

- Optional: Static graph

- Functions (not Sessions)

- Declarative programming model

- Regular Python functions

- **tf.function** decorator allows turning such functions into graphs (to execute remotely, serialize, optimize performance)

- Using **tf.Session** discouraged as of Tensorflow 2.0

You should use Release 2

You should use most-recent version  
that is available as **STABLE** release  
(release 2.11.1 or later stable version)

9

## Declarative (TF 1.x) vs. Imperative (TF 2.x)

### TF 1.14

```
import tensorflow as tf
x = tf.constant(1)
y = tf.constant(2)
z = tf.add(x, y)
print(z)
with tf.Session() as sess:
    print(sess.run(z))
```

Prints

You should use Release 2

You should use most-recent version  
that is available as **STABLE** release  
(release 2.11.1 or later stable version)

Tensor("Add:0", shape=(), dtype=int32)

Prints

3

### TF 2.x

```
import tensorflow as tf
x = tf.constant(1)
y = tf.constant(2)
z = x + y
print(z)
```

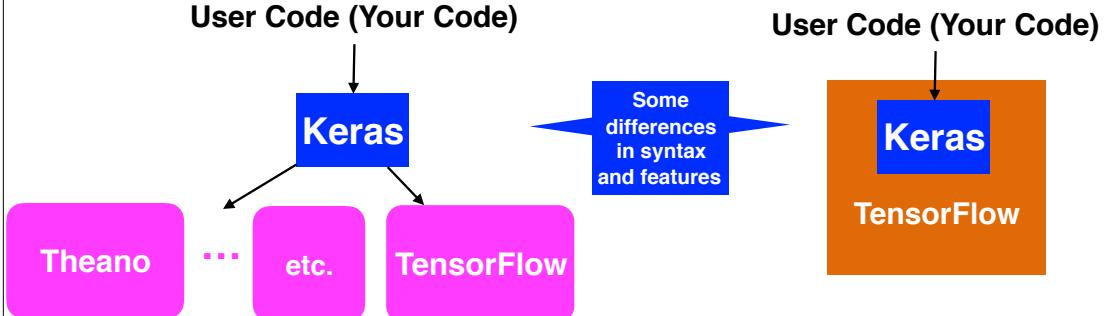
Prints

Evaluated immediately

tf.Tensor(3, shape=(), dtype=int32)

# Keras: High-Level Deep Learning API

- Introduced by F. Chollet in 2015
  - Abstracts ANN components at high level — e.g., MLP layers



- Multi-backend Keras
  - Portable interface between user code and multiple DL libraries
  - Including as “contributed package” to TF
- For TensorFlow:
  - As of TF 2.0 TF Keras is part of TF, e.g.:  
from tensorflow.keras import layers

## TF 2.x vs. “Legacy” TF 1.x

- TF 2.x differs SIGNIFICANTLY from TF 1.x
  - New features
  - Changes, removals, etc.
- Examples of differences between TF 2.0 vs TF 1.x differences:
  - API changes — symbol changes, renames, removals, argument names changes, clean-up
    - ♦ E.g., removal of `tf.app`, `tf.flags`, and `tf.logging`
  - Unification of `tf.train.Optimizers` and `tf.keras.Optimizers`
    - ♦ Use `tf.keras.Optimizers` (and `GradientTape`) to compute gradients
  - `compute_gradients` removed as public API
  - Unification of exchange formats
    - ♦ Model state save-to and restore-from `SavedModel`
      - `SavedModels_accepted` by TensorFlow Lite, TensorFlow JS, TensorFlow Serving, ...
- See “TensorFlow DOT org”
- Also related: “GitHub DOT com SLASH tensorflow SLASH tensorflow”

## TF 2.x vs. “Legacy” TF 1.x (cont.)

- Examples of differences between TF 2.0 vs TF 1.x. (cont.):
  - Removed methods for global variable
    - E.g., `tf.global_variables_initializer`, `tf.get_global_step`
  - `AutoGraph`
    - Translate Python control flow into TF expressions
    - Allows regular Python inside `tf.function`-decorated functions
    - Also applied in functions used with `tf.data`, `tf.distribute` and `tf.keras` APIs
- Compatibility (and incompatibility) of TF\_2.x with TF\_1.x
  - `compat.v1` module for compatibility with TF\_1.x
  - Multiple TF\_2.x features INCOMPATIBLE with TF\_1.x
    - Will not run under TF 2.x !

You should use Release 2  
You should use most-recent version  
that is available as **STABLE** release  
(release 2.11.1 or later stable version)

- See “TensorFlow DOT org”
- Also related: “GitHub DOT com SLASH tensorflow SLASH tensorflow”

## TensorFlow Datasets – TFDS

TF 2.9.1

```
import tensorflow as tf
import tensorflow_datasets as tfds

tf.__version__      → '2.9.1'
tfds.__version__   → '4.6.0'

dataset = tfds.load(name='.....')
```

Dataset name  
• See names of avaialble datasets on tensorflow DOT org

→ Downloading and preparing dataset

...  
...

## **Questions?**

**We will continue discussing TensorFlow  
in subsequent lecture/lectures**

---

# **Neural Networks and Deep Learning**

**Dr. Jerome J. Braun**

## **This Lecture: PyTorch Essentials I**

Course: Neural Networks and Deep Learning  
IE 7615

---

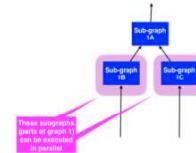
J. Braun

---

**Unauthorized distributing, redistributing, posting,  
and/or reposting, of the materials of this course  
(including course lectures, lecture slides, slide-sets,  
syllabi, guidelines, assignments, quizzes, exams,  
presentations, software, electronic media, etc.),  
is prohibited.**

# PyTorch Overview

- PyTorch – very popular in academia
  - Academic research related to deep neural networks
- “Look and feel” similar to NumPy
- Straightforward support of CUDA GPUs (NVIDIA)
- Computational graphs
  - Dynamic in PyTorch
- Simplicity and ease of use



See “pytorch DOT org”

# PyTorch Overview (cont.)

## Multiple versions of PyTorch available

- Some v0.x.x versions still available — should not be used
- Versions 1.x.x
  - v1.13.1 — current stable release
  - v1.0.0 through v1.13.1
- Potentially-unstable versions
  - Master (unstable)

Most current **stable** release  
routinely recommended

If you use GPUs you may need one of  
these versions, dependent on your  
CUDA version

- Most PyTorch versions are available  
only for specific CUDA versions

# PyTorch Overview (cont.)

- **torch.Tensor**
  - Similar to numpy array
- **torch.autograd**
  - Building computational graphs
  - Automatic computation of gradients
- **torch.nn**
  - Neural network components, layers, etc.
  - Can manipulate their state, update weights, etc.
- **torch.optim**
  - Contains optimizers

See “pytorch DOT org”

J. Braun

5

## Installing PyTorch

Assumed:

- You have installed python
- You set up virtual environment as recommended earlier in this course
  - At least basic one such as venv
- Currently in virtual environment
- Numpy and matplotlib installed prior to installing PyTorch

In Linux or macOS:

```
python3 -m pip install --upgrade torch  
python3.x -m pip install --upgrade torch
```

If you have multiple versions of python: 3.x is version number (e.g., 3.9)

6

# PyTorch Tensors

```
import torch
```

Example

```
data = [  
    [0, 1],  
    [2, 3],  
    [4, 5]  
]  
x = torch.tensor(data)  
print(x)
```

Prints

```
tensor([[0, 1],  
        [2, 3],  
        [4, 5]])
```

```
x_float = torch.tensor(data, dtype=torch.float)  
print(x_float)
```

Prints

```
tensor([[0., 1.],  
        [2., 3.],  
        [4., 5.]])
```

```
z = torch.randn(N, M)
```

J. Braun

7

## Domain-specific Components of PyTorch

- Domain-specific libraries of datasets
  - TorchVision
    - ♦ Image data
  - TorchText
    - ♦ Text data
  - TorchAudio
    - ♦ Audio data
- **torchvision.datasets module**
  - Dataset objects for image data, e.g., MNIST, CIFAR, COCO, etc.

See “pytorch DOT org”

J. Braun

8

# PyTorch Datasets and Transforms

```
from torchvision import datasets  
mnist_train =  
    datasets.MNIST('data', train=True, download=True)  
...  
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz  
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to data/MNIST/raw/train-  
images-idx3-ubyte.gz  
9913344/?|00:21<00:00. 451740.52it/s
```

TorchVision example:  
MNIST dataset

```
import matplotlib.pyplot as plt  
for k, (image, label) in enumerate(mnist_train[:6]):  
    plt.subplot(2, 3, k+1) plt.imshow(image, cmap='gray')
```



```
from torchvision import transforms
```

transforms module —  
various data-manipulation  
operations

J. Braun

9

# torch.autograd

- PyTorch implementation of autodiff

```
import torch  
x = torch.tensor(-2.0, requires_grad=True)
```

Track operations  
involving this tensor

```
...  
f = Q(x,y,z,...)  
...  
f.backward()
```

Hence f depends on x

```
print(x.grad)
```

Prints value of df/dx

J. Braun

10

# torch.nn

- Building blocks for neural networks
  - From basic to advanced

```
import torch
N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)
model = torch.nn.Sequential(
    torch.nn.Linear(D_in, H),
    torch.nn.ReLU(),
    torch.nn.Linear(H, D_out))
...
```

Example

J. Braun

11

# Running PyTorch on GPU

- Easy use of CUDA (NVIDIA) GPU devices

```
import torch
if torch.cuda.is_available():
    deviceNow = 'cuda'
else:
    deviceNow = 'cpu'
...
x = torch.randn(2, device=deviceNow)
...
y = torch.randn(2).to('cuda')
```

Allocate on GPU

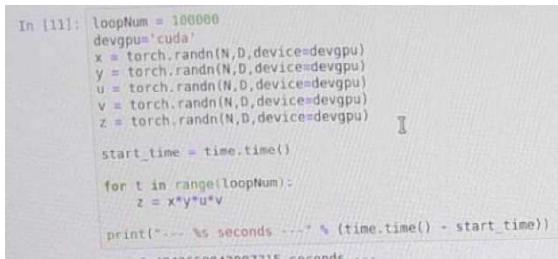
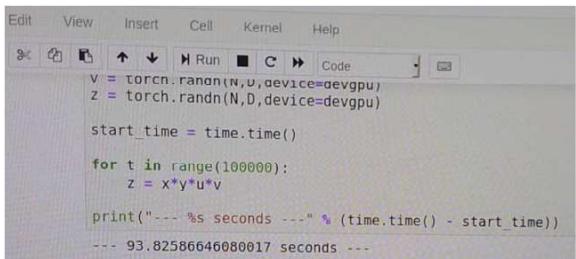
Transfer to GPU

J. Braun

12

# Running on GPU

Both TensorFlow and PyTorch have GPU support



```
loopNum = 100000
devgpu='cuda'
x = torch.randn(N,D,device=devgpu)
y = torch.randn(N,D,device=devgpu)
u = torch.randn(N,D,device=devgpu)
v = torch.randn(N,D,device=devgpu)
z = torch.randn(N,D,device=devgpu)

start_time = time.time()

for t in range(loopNum):
    z = x*y*u*v

print("--- %s seconds ---" % (time.time() - start_time))
--- 93.82586646080017 seconds ---
```

On CPU: 93.8 sec.      ← same test loop →      On GPU: 6.4 sec



Pay attention to GPU temperature

J. Braun

13

## Questions?

J. Braun

14

---

---

# **Neural Networks and Deep Learning**

**Dr. Jerome J. Braun**

## **This Lecture:**

**Computational Graphs and Back-Propagation in Deep Feedforward MLP Networks I**

Course: Neural Networks and Deep Learning  
IE 7615

---

J. Braun

---

---

**Unauthorized distributing, redistributing, posting,  
and/or reposting, of the materials of this course  
(including course lectures, lecture slides, slide-sets,  
syllabi, guidelines, assignments, quizzes, exams,  
presentations, software, electronic media, etc.),  
is prohibited.**

# This Lecture

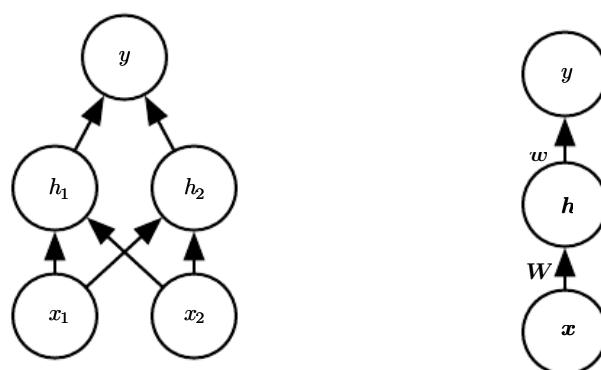
---



- More on computational graphs
- Computational graphs and back-propagation for deep feedforward MLP networks (part 1)

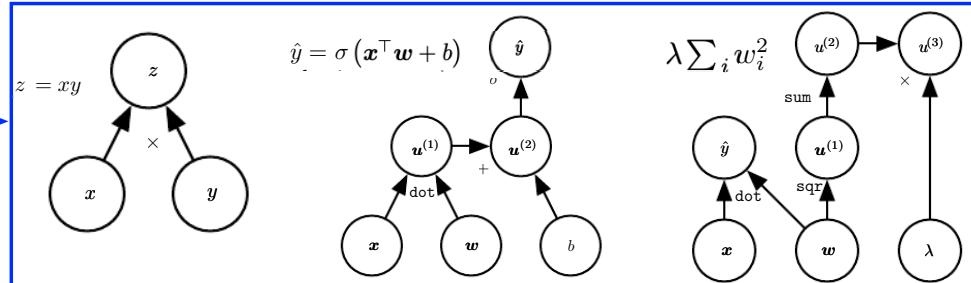
## Recap: Feedforward ANN Graphical Representation

---



# Computational Graph Examples

Recap:

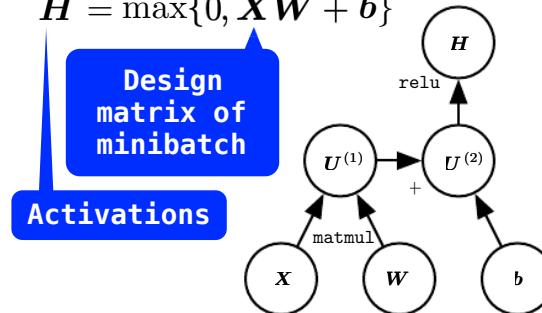


[Goodfellow, 2016]

$$H = \max\{0, \mathbf{X}\mathbf{W} + \mathbf{b}\}$$

Design matrix of minibatch

Activations



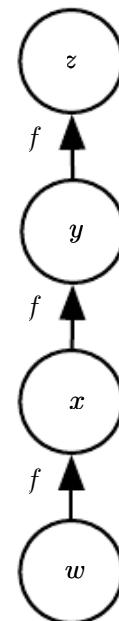
5

# Chain Rule and Computational Graph

$$f : \mathbb{R} \rightarrow \mathbb{R}$$

$$x = f(w), y = f(x), z = f(y)$$

$$\begin{aligned} & \frac{\partial z}{\partial w} \\ &= \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \frac{\partial x}{\partial w} \\ &= f'(y) f'(x) f'(w) \\ &= f'(f(f(w))) f'(f(w)) f'(w) \end{aligned}$$



6

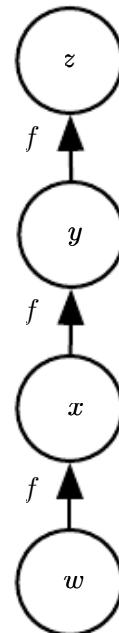
# Computing Gradient

$$x = f(w), y = f(x), z = f(y)$$

$$\begin{aligned}\frac{\partial z}{\partial w} \\ = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \frac{\partial x}{\partial w} \\ = f'(y) f'(x) f'(w)\end{aligned}$$

Backpropagation approach

- $f(w)$  computed ONCE
- Stored in variable  $x$



- Advantage of reduced runtime
- Preferable if memory required to store subexpressions is sufficiently low

7

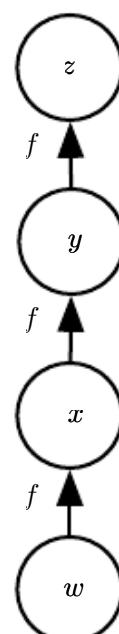
# Computing Gradient (cont.)

$$\begin{aligned}\frac{\partial z}{\partial w} \\ = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \frac{\partial x}{\partial w} \\ = f'(y) f'(x) f'(w) \\ = f'(f(f(w))) f'(f(w)) f'(w)\end{aligned}$$

Alternative approach

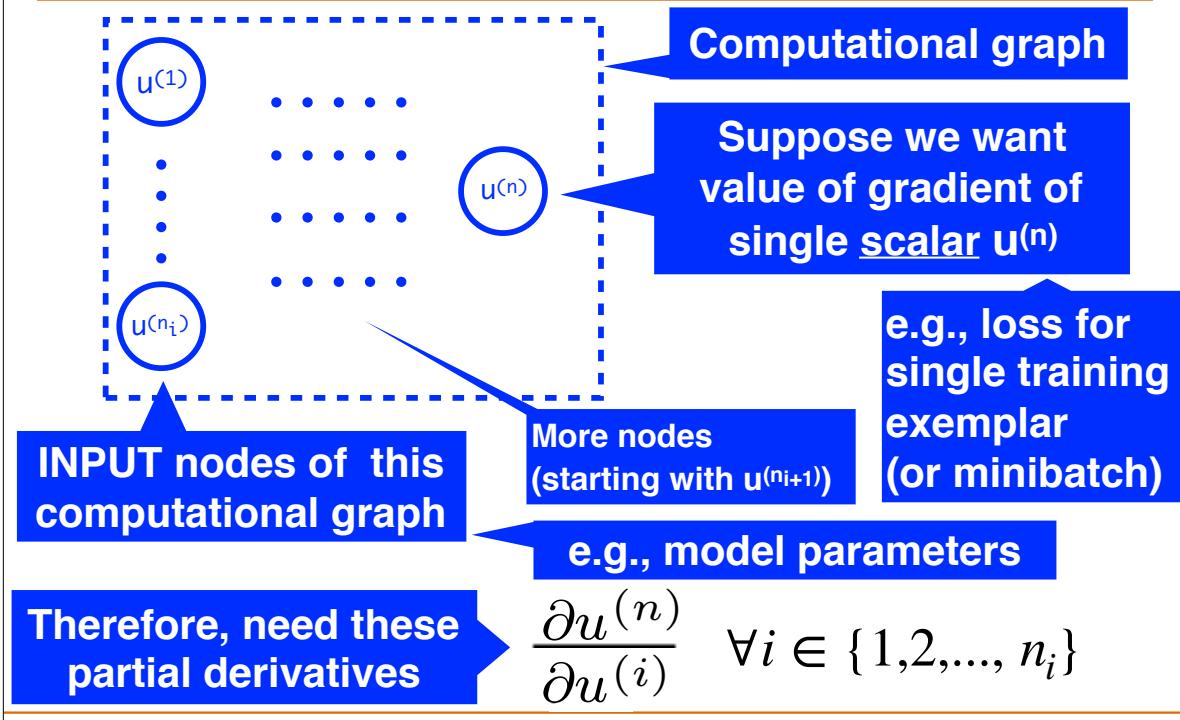
Repeated subexpressions

- Recompute these subexpression whenever needed
- Can be useful alternative if memory is limited



8

## Computational Graph to Compute Single Scalar $u^{(n)}$



J. Braun

9

## Computational Graph to Compute Single Scalar $u^{(n)}$

Assume node ordering such that

- Outputs of  $u^{(i)}$  can be computed in sequence

- Start at  $u^{(n_{i+1})}$
- Go up to  $u^{(i)}$

- Each node  $u^{(i)}$  associated with operation  $f^{(i)}$
- $A^{(i)}$  set of values of all nodes that are PARENTS of  $u^{(i)}$

$$u^{(i)} = f(\mathbb{A}^{(i)})$$

$$\frac{\partial u^{(n)}}{\partial u^{(j)}} = \sum_{i:j \in Pa(u^{(i)})} \frac{\partial u^{(n)}}{\partial u^{(i)}} \frac{\partial u^{(i)}}{\partial u^{(j)}}$$

10

## Pseudocode of Forward Propagation Graph to Map $u^{(n_i)}$ inputs to Output $u^{(n)}$

```
Graph to Map  $u^{(n_i)}$  inputs to Output  $u^{(n)}$ 
for  $i = 1, \dots, n_i$  do
     $u^{(i)} \leftarrow x_i$ 
end for
for  $i = n_i + 1, \dots, n$  do
     $\mathbb{A}^{(i)} \leftarrow \{u^{(j)} \mid j \in Pa(u^{(i)})\}$ 
     $u^{(i)} \leftarrow f^{(i)}(\mathbb{A}^{(i)})$ 
end for
return  $u^{(n)}$ 
```

[Goodfellow, 2016]

11

## Pseudocode of Back-propagation Graph to Map $u^{(n_i)}$ inputs to Output $u^{(n)}$

```
First run forward propagation to obtain network activations
Then:
```

Initialize `grad_table`, a data structure that will store the derivatives that have been computed. The entry `grad_table[u(i)]` will store the computed value of  $\frac{\partial u^{(n)}}{\partial u^{(i)}}$ .

```
grad_table[∂u(n)] ← 1
for  $j = n - 1$  down to 1 do
    The next line computes  $\frac{\partial u^{(n)}}{\partial u^{(j)}} = \sum_{i:j \in Pa(u^{(i)})} \frac{\partial u^{(n)}}{\partial u^{(i)}} \frac{\partial u^{(i)}}{\partial u^{(j)}}$  using stored values:
    grad_table[u(j)] ← ∑i:j ∈ Pa(u(i)) grad_table[u(i)] ∂u(i) / ∂u(j)
end for
return {grad_table[u(i)] | i = 1, …, n_i}
```

[Goodfellow, 2016]

12

# This Lecture

---

- More on computational graphs
- • Computational graphs and back-propagation for deep feedforward MLP networks (part 1)

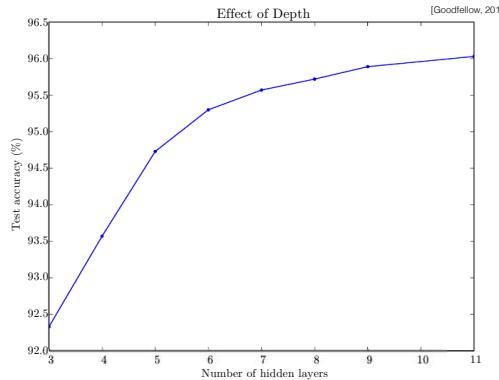
## Recap: Network Depth

---

- Universal Approximation Theorem
  - One hidden layer sufficient to approximate ANY function  $F$  to ARBITRARY degree of accuracy
- But:
  - Universal approximation theorem does NOT imply ability to LEARN the function, i.e., given any function  $F$ 
    - Single hidden-layer network approximator which represents any desired function  $F$  **does** exist
    - But learning algorithm may not be able to learn (i.e., find) it
    - Problem of hidden-layer size (width)

Rationale  
for  
Deep  
Learning

# Back-propagation in Deep Networks



- How to compute gradients (reliably and efficiently) when
  - Network is deep
  - Network has large (huge) number of parameters

15

## Chain Rule and Jacobian

Recall

$$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$$

Let  $\mathbf{x} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^n$

$\mathbf{y} = g(\mathbf{x})$  g maps from  $\mathbb{R}^m$  to  $\mathbb{R}^n$

$z = f(\mathbf{y})$  f maps from  $\mathbb{R}^n$  to R

Then  $\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$

i.e.,  $\nabla_{\mathbf{x}} z = \left( \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \right)^T \nabla_{\mathbf{y}} z,$

Jacobian matrix of g

n × m matrix

# Jacobian and Back-Propagation

$$\begin{aligned} \mathbf{x} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^n \quad & \mathbf{y} = g(\mathbf{x}) && \text{g maps from } \mathbb{R}^m \text{ to } \mathbb{R}^n \\ & z = f(\mathbf{y}) && \text{f maps from } \mathbb{R}^n \text{ to } \mathbb{R} \\ & \nabla_{\mathbf{x}} z = \left( \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \right)^{\top} \nabla_{\mathbf{y}} z && \end{aligned}$$

**Jacobian matrix of g**      **Gradient**

**Back-propagation:**

- Perform such Jacobian  $\times$  Gradient product for each operation in computational graph

# Generalization to Tensors

Consider tensors  $\mathbf{X}$  and  $\mathbf{Y}$ ,  
such that:

$$\mathbf{Y} = g(\mathbf{X})$$

$$z = f(\mathbf{Y})$$

Let index  $i$  represent entire tuple of tensor indices

Then →

$$(\nabla_{\mathbf{X}} z)_i \quad \frac{\partial z}{\partial X_i}$$

Gradient  
w.r.t.  
tensor  $\mathbf{X}$

$$\nabla_{\mathbf{X}} z = \sum_j (\nabla_{\mathbf{X}} Y_j) \frac{\partial z}{\partial Y_j}$$

## Pseudocode of Forward Propagation thru Deep NN

[Goodfellow, 2016]

```

Require: Network depth,  $l$ 
Require:  $\mathbf{W}^{(i)}, i \in \{1, \dots, l\}$ , the weight matrices of the model
Require:  $\mathbf{b}^{(i)}, i \in \{1, \dots, l\}$ , the bias parameters of the model
Require:  $\mathbf{x}$ , the input to process
Require:  $\mathbf{y}$ , the target output
 $\mathbf{h}^{(0)} = \mathbf{x}$ 
for  $k = 1, \dots, l$  do
     $\mathbf{a}^{(k)} = \mathbf{b}^{(k)} + \mathbf{W}^{(k)} \mathbf{h}^{(k-1)}$ 
     $\mathbf{h}^{(k)} = f(\mathbf{a}^{(k)})$ 
end for
 $\hat{\mathbf{y}} = \mathbf{h}^{(l)}$ 
 $J = L(\hat{\mathbf{y}}, \mathbf{y}) + \lambda \Omega(\theta)$ 

```

Computing cost function

19

## Pseudocode of Backward Propagation thru Deep NN

Yields gradients on activations  $\mathbf{a}^{(k)}$  for each layer  $k$   
 From output layer to first hidden layer

[Goodfellow, 2016]

After the forward computation, compute the gradient on the output layer:

$\mathbf{g} \leftarrow \nabla_{\hat{\mathbf{y}}} J = \nabla_{\hat{\mathbf{y}}} L(\hat{\mathbf{y}}, \mathbf{y})$   
 for  $k = l, l-1, \dots, 1$  do

Convert the gradient on the layer's output into a gradient into the pre-nonlinearity activation (element-wise multiplication if  $f$  is element-wise):

$$\mathbf{g} \leftarrow \nabla_{\mathbf{a}^{(k)}} J = \mathbf{g} \odot f'(\mathbf{a}^{(k)})$$

Compute gradients on weights and biases (including the regularization term, where needed):

$$\begin{aligned}\nabla_{\mathbf{b}^{(k)}} J &= \mathbf{g} + \lambda \nabla_{\mathbf{b}^{(k)}} \Omega(\theta) \\ \nabla_{\mathbf{W}^{(k)}} J &= \mathbf{g} \mathbf{h}^{(k-1)\top} + \lambda \nabla_{\mathbf{W}^{(k)}} \Omega(\theta)\end{aligned}$$

Propagate the gradients w.r.t. the next lower-level hidden layer's activations:

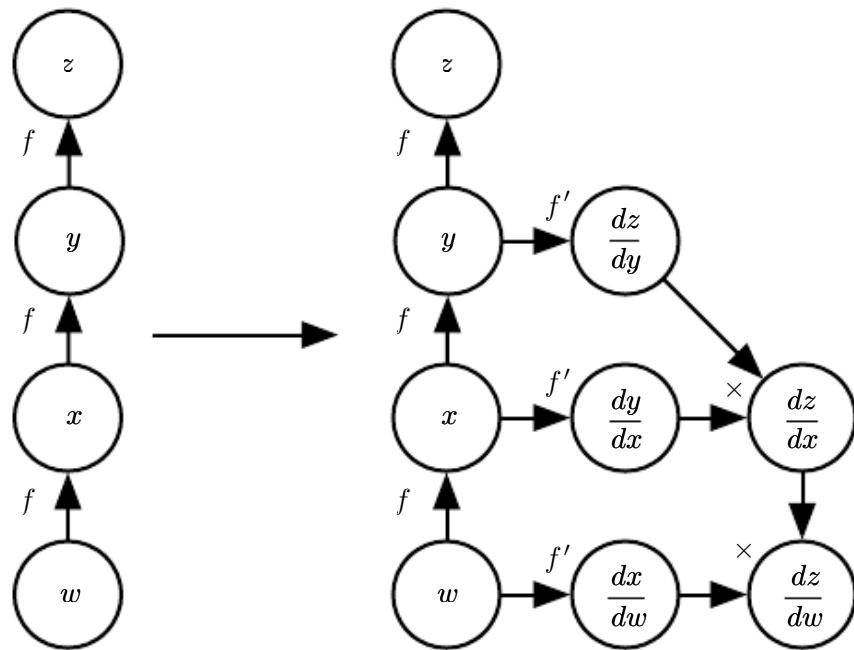
$$\mathbf{g} \leftarrow \nabla_{\mathbf{h}^{(k-1)}} J = \mathbf{W}^{(k)\top} \mathbf{g}$$

end for

Gradient on parameters for each layer

20

# Symbol-to-Symbol Differentiation



21

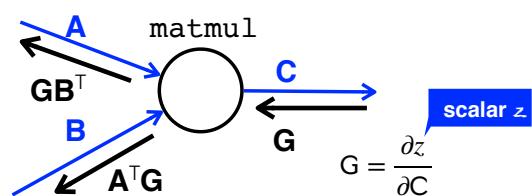
## op.bprop

Each operation **Op** is associated with **bprop** operation

**bprop** computes Jacobian-vector product  $\nabla_{\mathbf{X}} z = \sum_j (\nabla_{\mathbf{X}} Y_j) \frac{\partial z}{\partial Y_j}$

Each operation is responsible ONLY for ability to back-propagate through graph-edges in which it participates

E.g., consider  $C=AB$   
(A,B,C are tensors)



22

## op.bprop(inputs, X, G)

$$\nabla_X z = \sum_j (\nabla_X Y_j) \frac{\partial z}{\partial Y_j}$$

Mathematical function that operation implements

$$\sum_i (\nabla_X op.f(inputs)_i) G_i$$

Input whose gradient is wanted

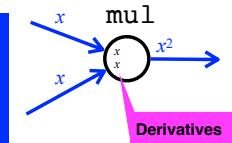
Gradient on output of operation

List of inputs supplied to operation

op.bprop method should always “pretend” all its inputs distinct from each other (even if inputs identical)

Example: mul operator to compute  $x^2$

- op.bprop passed two copies of x as inputs
- op.bprop should return x as derivative w.r.t. to both inputs
- Back-propagation algorithm will later add them together to obtain  $2x$



Next will look at back-propagation pseudocode

23

## Functions for Back-propagation Pseudocode

`get_operation(V)`

Returns operation that computes variable V, represented by edges coming into node V

Example: Matrix multiplication  $C = AB$ . Assume class “matmul”

`get_operation(V)` returns pointer to instance of “matmul”

`get_consumers(V, G)`

Returns list of variables that are children of V in G

`get_inputs(V, G)`

Returns list of variables that are parents of V in G

24

## Back-Propagation Pseudocode — Outer Loop

### Setup and cleanup

[Goodfellow, 2016]

Require:  $\mathbb{T}$ , the target set of variables whose gradients must be computed.  
Require:  $\mathcal{G}$ , the computational graph  
Require:  $z$ , the variable to be differentiated  
Let  $\mathcal{G}'$  be  $\mathcal{G}$  pruned to contain only nodes that are ancestors of  $z$  and descendants of nodes in  $\mathbb{T}$ .  
Initialize `grad_table`, a data structure associating tensors to their gradients  
`grad_table`[ $z$ ]  $\leftarrow 1$   
**for**  $V$  in  $\mathbb{T}$  **do**  
    `build_grad`( $V, \mathcal{G}, \mathcal{G}', \text{grad\_table}$ )  
**end for**  
Return `grad_table` restricted to  $\mathbb{T}$

Main part: inner loop  
(will see next)

25

## Back-Propagation Pseudocode — Inner Loop

### Pseudocode of `Build_grad(V, G, G', grad_table)` function:

Require:  $V$ , the variable whose gradient should be added to  $\mathcal{G}$  and `grad_table`.  
Require:  $\mathcal{G}$ , the graph to modify.  
Require:  $\mathcal{G}'$ , the restriction of  $\mathcal{G}$  to nodes that participate in the gradient.  
Require: `grad_table`, a data structure mapping nodes to their gradients  
**if**  $V$  is in `grad_table` **then**  
    Return `grad_table`[ $V$ ]  
**end if**  
 $i \leftarrow 1$   
**for**  $C$  in `get_consumers`( $V, \mathcal{G}'$ ) **do**  
     $op \leftarrow \text{get\_operation}(C)$   
     $D \leftarrow \text{build_grad}(C, \mathcal{G}, \mathcal{G}', \text{grad\_table})$   
     $G^{(i)} \leftarrow op.bprop(\text{get\_inputs}(C, \mathcal{G}'), V, D)$   
     $i \leftarrow i + 1$   
**end for**  
 $G \leftarrow \sum_i G^{(i)}$   
`grad_table`[ $V$ ] =  $G$   
Insert  $G$  and the operations creating it into  $\mathcal{G}$   
Return  $G$

[Goodfellow, 2016]

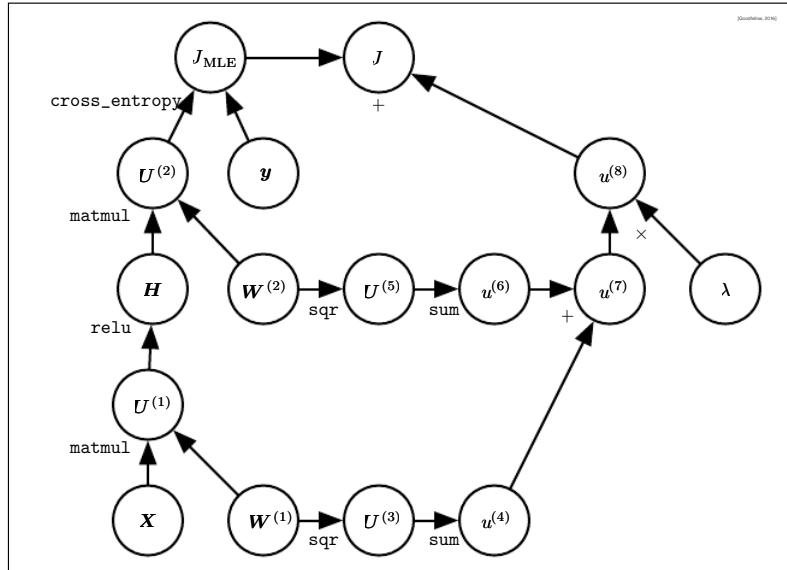
26

## Symbol-to-Symbol Differentiation and Back-propagation

**Example:**  
Single hidden-layer feedforward MLP network

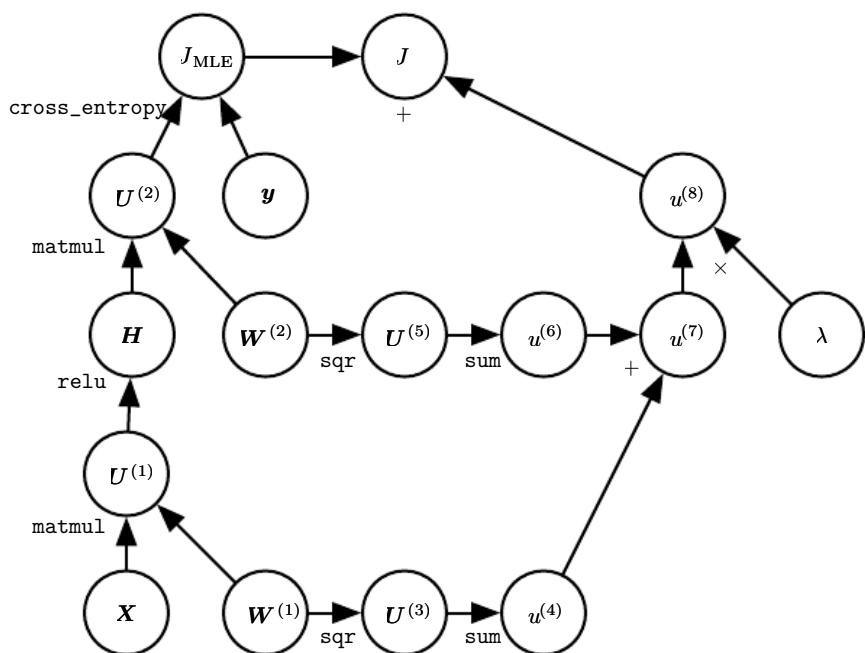
$$J = J_{\text{MLE}} + \lambda \left( \sum_{i,j} \left( W_{i,j}^{(1)} \right)^2 + \sum_{i,j} \left( W_{i,j}^{(2)} \right)^2 \right)$$

**Cost function:**  
cross-entropy,  
weight decay  
regularization



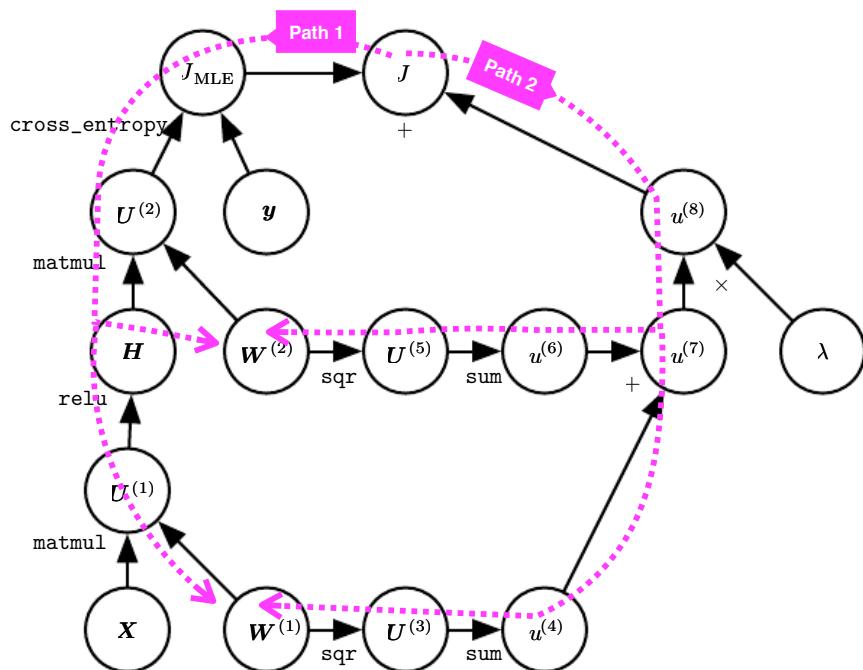
27

## Back-propagation Paths



28

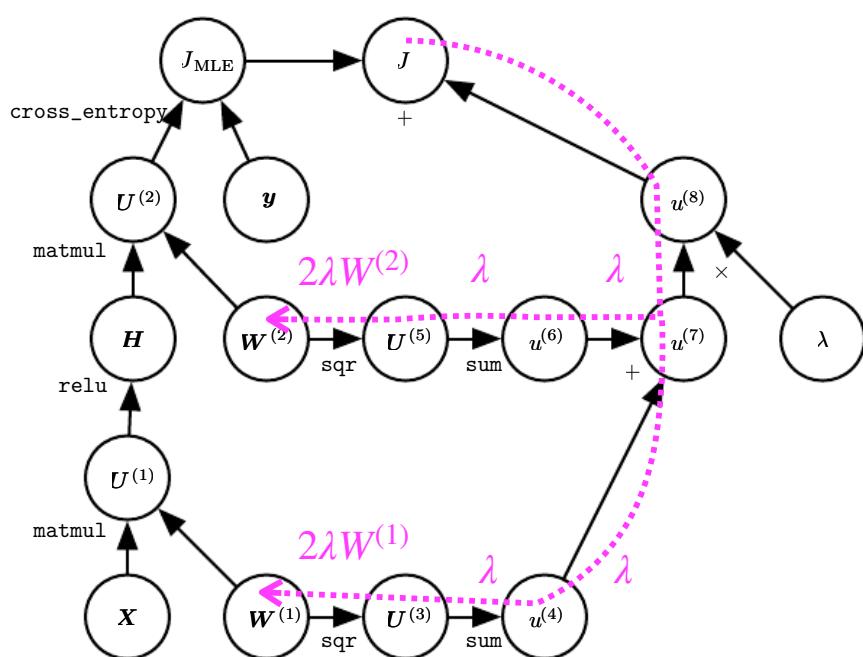
# Back-propagation Paths



J. Braun

29

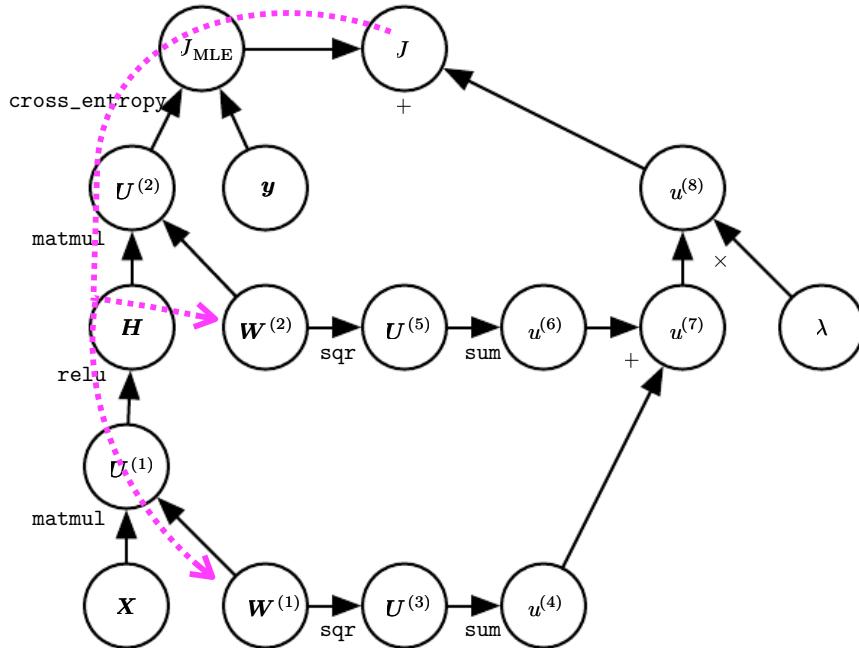
# Back-propagating Gradients



J. Braun

30

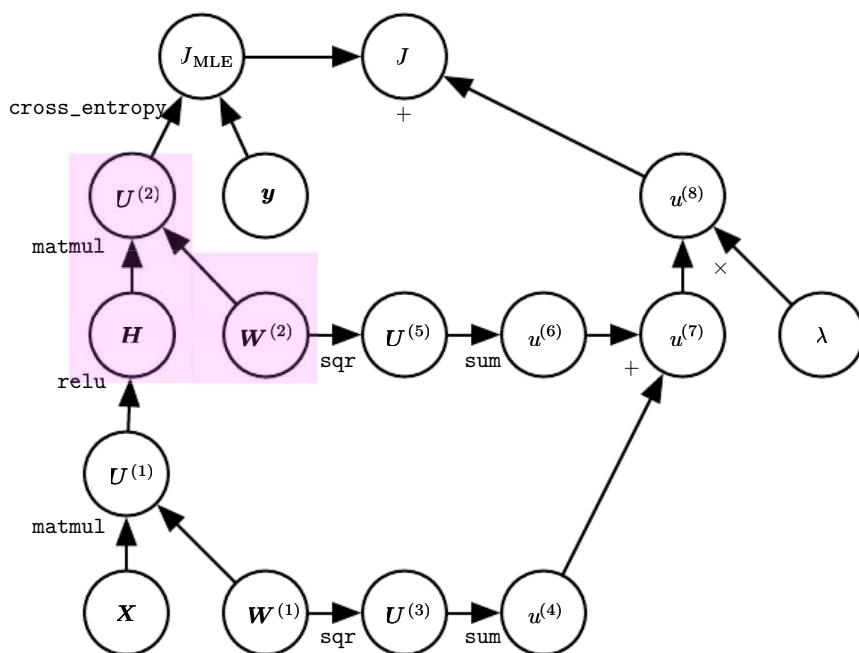
# Back-propagating Gradients



J. Braun

31

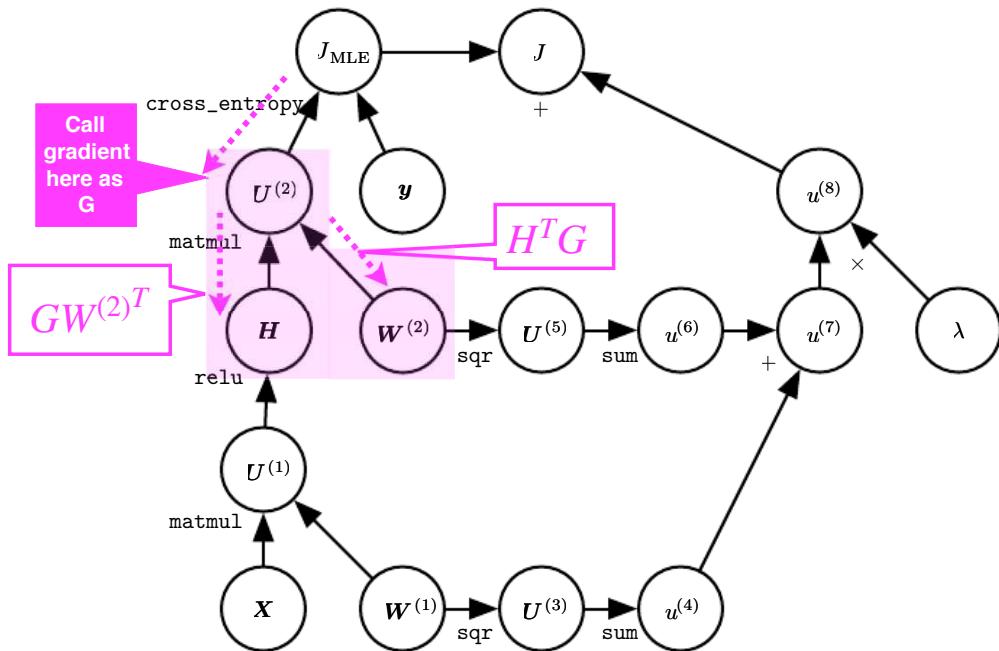
# Back-propagating Gradients



J. Braun

32

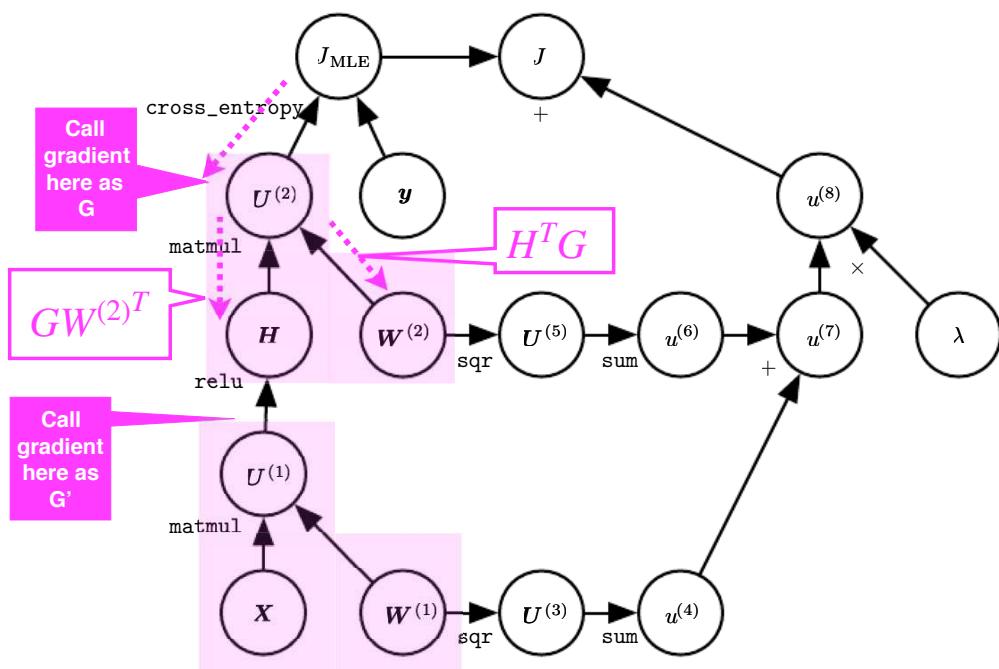
# Back-propagating Gradients



J. Braun

33

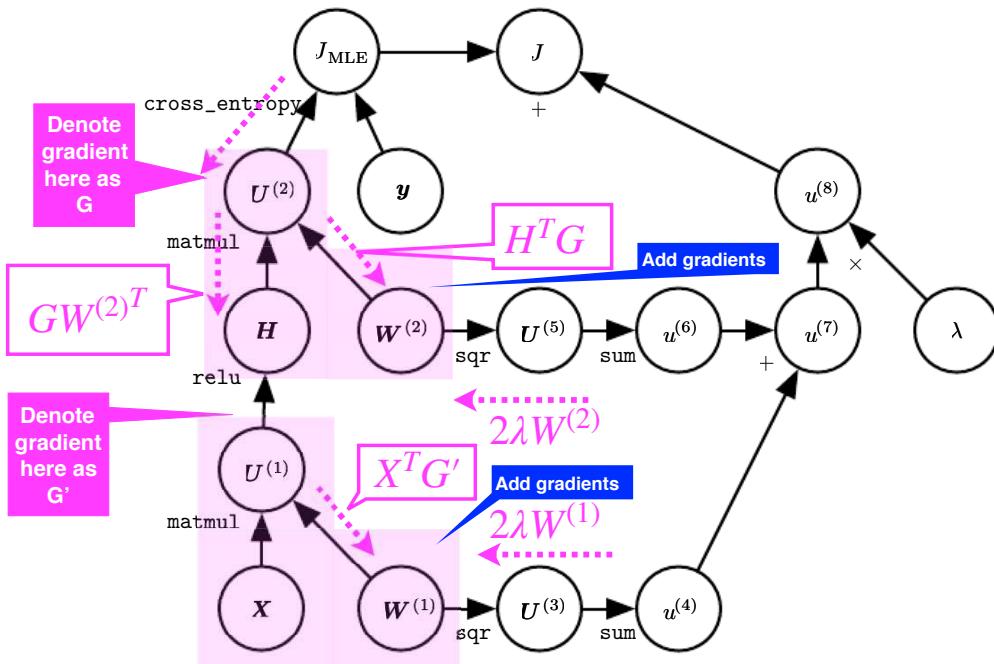
# Back-propagating Gradients



J. Braun

34

# Back-propagating Gradients



J. Braun

35

## Questions?

---

# **Neural Networks and Deep Learning**

**Dr. Jerome J. Braun**

## **This Lecture: Training Modes and Momentum**

Course: Neural Networks and Deep Learning  
IE 7615

---

J. Braun

---

**Unauthorized distributing, redistributing, posting,  
and/or reposting, of the materials of this course  
(including course lectures, lecture slides, slide-sets,  
syllabi, guidelines, assignments, quizzes, exams,  
presentations, software, electronic media, etc.),  
is prohibited.**

# Recap: Batch and On-Line Learning

---

- **Epoch**
  - Pass through all training patterns
    - ♦ Each presented to training algorithms
- **Batch learning**
  - Changes accumulated over all patterns
  - Model (weights) update after entire epoch
- **On-line learning**
  - Model (weights) update each pattern

## Minibatch

---

- **Minibatch training**
  - Partition training set into subsets (minibatches)
- **Model (weights) update after subset of training patterns**
  - Any pattern in exactly one minibatch
- **Typical for deep-learning algorithms**
- **In deep learning**
  - **Minibatch often referred to as “batch”**

# Minibatch Algorithms

---

- Larger batches provide more accurate estimate of gradient
  - But not linearly
- Multicore architectures usually underutilized if minibatches are extremely small
  - No time-reduction to process batch if below some absolute minimum batch-size
- If all exemplars in batch processed in parallel (typical)
  - Memory scales with batch size
    - ♦ Limiting factor in batch-size

# Minibatch Algorithms

---

- For some hardware architectures runtime better when using specific sizes of arrays
  - For GPUs, runtime better if using power-of-2 batch sizes
    - ♦ Typical values from 32 to 256
    - ♦ Sometimes 16 for large models
- Small batches can offer regularizing effect
  - Perhaps due to noise they add to learning process
  - Generalization error often best for batch-size of 1
    - ♦ But might require small learning-rate to maintain stability due to high variance in gradient-estimate
    - ♦ Total runtime can be very high
      - More steps due to more minibatches to cover entire training dataset
      - More steps due to reduced learning-rate

Reminder: in deep-learning parlance “batch” actually means minibatch

# Stochastic Gradient Descent (SGD)

- Extension of Gradient-Descent algorithm
  - Gradient descent with minibatches
- Crucial for deep learning
- Good generalization => large training datasets
- SGD motivation:
  - Processing large training datasets is computationally expensive

## Decomposing Cost Function – Example

$$L(\mathbf{x}, y, \boldsymbol{\theta}) = -\log p(y \mid \mathbf{x}; \boldsymbol{\theta})$$

Per-exemplar loss

Negative conditional log-likelihood of training data

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}, y \sim \hat{p}_{\text{data}}} L(\mathbf{x}, y, \boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^m L(\mathbf{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$$

Gradient computation for such additive cost functions

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^m \nabla_{\boldsymbol{\theta}} L(\mathbf{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$$

O(m) time complexity for each gradient computation

- Prohibitive for large datasets

# SGD and Minibatch

$$L(\mathbf{x}, y, \boldsymbol{\theta}) = -\log p(y \mid \mathbf{x}; \boldsymbol{\theta})$$

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}, y \sim \hat{p}_{\text{data}}} L(\mathbf{x}, y, \boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^m L(\mathbf{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$$

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^m \nabla_{\boldsymbol{\theta}} L(\mathbf{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$$

Gradient is expectation

Expectation can be estimated  
approximately from small  
number of samples (exemplars)

Key insight and idea behind SGD

J. Braun

9

# SGD and Minibatch

At each step sample minibatch of exemplars  
drawn uniformly from training dataset

$$\mathbb{B} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m')}\}$$

m' typically small  
• 1 to few hundreds

m' kept fixed as training dataset grows (e.g., to billions)

Can update using gradient estimate  
computed on few (m') exemplars

$$\mathbf{g} = \frac{1}{m'} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m'} L(\mathbf{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$$

Weight update for each minibatch  
step (downhill)

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \mathbf{g}$$

J. Braun

10

# SGD and Learning Rate

- Fixed learning-rate issue
  - SGD gradient estimator introduces noise that does not vanish near minima
    - Noise is due to sampling minibatches from all  $m$  exemplars of full dataset
      - This does not happen in (full) batch gradient descent (gradient tends to 0 near minima)
- In practice, gradually decrease learning rate

Change learning rate up to iteration  $\tau$ , then keep it constant at  $\epsilon_\tau$

$$\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha\epsilon_\tau$$
$$\alpha = \frac{k}{\tau}$$

SGD update at iteration  $k$ :

Require: Learning rate  $\epsilon_k$ .

Require: Initial parameter  $\theta$

while stopping criterion not met do

    Sample a minibatch of  $m$  examples from the training set  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  with corresponding targets  $\mathbf{y}^{(i)}$ .

    Compute gradient estimate:  $\hat{\mathbf{g}} \leftarrow +\frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

    Apply update:  $\theta \leftarrow \theta - \epsilon \hat{\mathbf{g}}$

end while

[Goodfellow, 2016]

11

# Momentum

- Recall idea behind momentum
  - Prevent “overshooting” minimum

Velocity

$$\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \nabla_{\theta} \left( \frac{1}{m} \sum_{i=1}^m L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)}) \right)$$
$$\theta \leftarrow \theta + \mathbf{v}$$

J. Braun

12

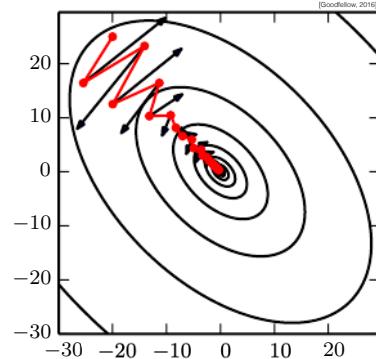
# Momentum

$$v \leftarrow \alpha v - \epsilon \nabla_{\theta} \left( \frac{1}{m} \sum_{i=1}^m L(f(x^{(i)}; \theta), y^{(i)}) \right)$$

$$\theta \leftarrow \theta + v$$

- Momentum aims at mitigating issues of

- Poor conditioning of Hessian matrix
- Variance in stochastic gradient



J. Braun

13

# Momentum

$$v \leftarrow \alpha v - \epsilon \nabla_{\theta} \left( \frac{1}{m} \sum_{i=1}^m L(f(x^{(i)}; \theta), y^{(i)}) \right) \quad \theta \leftarrow \theta + v$$

Step size depends on BOTH of these for SEQUENCE of gradients:

- How large gradients are AND
- How aligned they are

Step size largest when many SUCCESSIVE gradients point in EXACTLY same direction

If always observe (same) gradient  $g$ ,  
then will accelerate in direction  $-g$   
until reaching terminal velocity

where step size is:

$$\frac{\epsilon ||g||}{1 - \alpha}$$

J. Braun

14

# Momentum

$$v \leftarrow \alpha v - \epsilon \nabla_{\theta} \left( \frac{1}{m} \sum_{i=1}^m L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)}) \right) \quad \theta \leftarrow \theta + v$$

Step size  $\frac{\epsilon \|\mathbf{g}\|}{1 - \alpha}$

Common practice values  
 $\alpha = 0.5, 0.9, 0.99$

## SGD with momentum

Require: Learning rate  $\epsilon$ , momentum parameter  $\alpha$ .

Require: Initial parameter  $\theta$ , initial velocity  $v$ .

while stopping criterion not met do

    Sample a minibatch of  $m$  examples from the training set  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  with corresponding targets  $\mathbf{y}^{(i)}$ .

    Compute gradient estimate:  $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

    Compute velocity update:  $v \leftarrow \alpha v - \epsilon g$

    Apply update:  $\theta \leftarrow \theta + v$

end while

[Goodfellow, 2016]

J. Braun

15

# Nesterov Momentum

$$v \leftarrow \alpha v - \epsilon \nabla_{\theta} \left[ \frac{1}{m} \sum_{i=1}^m L\left(f(\mathbf{x}^{(i)}; \theta + \alpha v), \mathbf{y}^{(i)}\right) \right] \quad \theta \leftarrow \theta + v$$

How it differs from standard momentum

Gradient evaluated AFTER applying current velocity

Can view as applying correction factor to standard momentum

## SGD with Nesterov momentum

Require: Learning rate  $\epsilon$ , momentum parameter  $\alpha$ .

Require: Initial parameter  $\theta$ , initial velocity  $v$ .

while stopping criterion not met do

    Sample a minibatch of  $m$  examples from the training set  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  with corresponding labels  $\mathbf{y}^{(i)}$ .

    Apply interim update:  $\tilde{\theta} \leftarrow \theta + \alpha v$

    Compute gradient (at interim point):  $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\theta}} \sum_i L(f(\mathbf{x}^{(i)}; \tilde{\theta}), \mathbf{y}^{(i)})$

    Compute velocity update:  $v \leftarrow \alpha v - \epsilon g$

    Apply update:  $\theta \leftarrow \theta + v$

end while

[Goodfellow, 2016]

J. Braun

16

## Nesterov Momentum and Rate of Convergence

$$v \leftarrow \alpha v - \epsilon \nabla_{\theta} \left[ \frac{1}{m} \sum_{i=1}^m L(f(x^{(i)}; \theta + \alpha v), y^{(i)}) \right] \quad \theta \leftarrow \theta + v$$

How it differs from standard momentum

Gradient evaluated AFTER applying current velocity

In convex batch gradient descent, Nesterov momentum:

- Rate of convergence of excess error after k iterations reduced from  $O(1/k)$  to  $O(1/k^2)$

In stochastic gradient descent, Nesterov momentum does not improve rate of convergence

## Questions?

---

# **Neural Networks and Deep Learning**

**Dr. Jerome J. Braun**

## **This Lecture: Adaptive Learning-Rate Algorithms**

Course: Neural Networks and Deep Learning  
IE 7615

---

J. Braun

---

**Unauthorized distributing, redistributing, posting,  
and/or reposting, of the materials of this course  
(including course lectures, lecture slides, slide-sets,  
syllabi, guidelines, assignments, quizzes, exams,  
presentations, software, electronic media, etc.),  
is prohibited.**

# This Lecture

---

- AdaGrad (and AdaDelta)
- RMSProp
- RMSProp with Nesterov Momentum
- Adam

# Learning-Rate Problem

---

- Selecting and adjusting learning-rate difficult
  - Significant impact on model performance
- Learning rate
  - One of most difficult hyperparameters to set/adjust
  - Very significant impact on model performance
  - Cost sensitivity to directions in parameter space
    - ♦ Highly variable — sensitive in some but not in other directions
  - Momentum can mitigate somewhat
    - ♦ But introduces another hyperparameter to set

# Adaptive Learning-Rates

- Separate learning-rates for each model-parameter
- Automatically adapt learning-rates throughout training
- Delta-bar-delta algorithm — early heuristic approach to adapting individual learning rates (Jacobs, 1988)
  - Learning rate for any given parameter depends on sign of partial derivative of loss function w.r.t. that parameter
    - If sign does not change, learning rate increases
    - If sign changes, learning rate decreases
  - Such rule applicable to batch training only — not to minibatch
- Modern adaptive learning-rates methods
  - Minibatch based

J. Braun

5

# AdaGrad

- AdaGrad algorithm (Duchi, 2011)
- Learning rates for all model parameters adapted individually
  - Scaled in inverse proportion to square root of sum of accumulated (historical) squared values
  - Learning rates of model parameters with largest partial derivative of loss function decrease rapidly
  - Learning rates of model parameters with small partial derivative of loss function decrease relatively slowly
- Leads to faster progress along gently-sloped directions of parameter space
- For convex optimization
  - AdaGrad algorithm has desirable theoretical properties
- AdaGrad in practice for training deep neural networks
  - AdaGrad may lead to premature and excessive decrease of effective learning rate
  - Performs well for some, not all, models

J. Braun

6

## AdaDelta

- AdaDelta (Zeiler, 2012) – extension of Adagrad
- Less aggressive decrease of learning rate
- Instead of accumulating all past squared gradients, use fixed-size window of past gradients accumulation
- Sum of gradients defined recursively as decaying average of all past squared gradients

J. Braun

7

## AdaGrad

Require: Global learning rate  $\epsilon$

Require: Initial parameter  $\theta$

Require: Small constant  $\delta$ , perhaps  $10^{-7}$ , for numerical stability

Initialize gradient accumulation variable  $r = \mathbf{0}$

while stopping criterion not met do

    Sample a minibatch of  $m$  examples from the training set  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  with corresponding targets  $\mathbf{y}^{(i)}$ .

    Compute gradient:  $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

    Accumulate squared gradient:  $\mathbf{r} \leftarrow \mathbf{r} + \mathbf{g} \odot \mathbf{g}$

    Compute update:  $\Delta \theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \mathbf{g}$ . (Division and square root applied element-wise)

    Apply update:  $\theta \leftarrow \theta + \Delta \theta$

end while

(Sandhesh, 2016)

8

# AdaGrad in Non-Convex Regions

---

- AdaGrad designed to converge rapidly for convex functions
- For non-convex loss functions (as in deep networks)
  - Learning trajectory may traverse multiple loss-function areas before reaching locally-convex area (bowl)
  - But since learning-rate is reduced according to *entire* history of squared gradient
    - Learning rate may be too small before reaching convex region

# RMSProp

---

- RMSProp algorithm (Hinton 2012) — modifies AdaGrad
  - Perform better in non-convex cases
  - Change gradient-accumulation to exponentially-weighted moving average
  - Exponential decay of average to discard extreme-past gradient-history
    - To converge fast upon reaching when convex region (bowl) found
  - Use of moving average introduces new hyperparameter  $\rho$ 
    - Control length-scale of moving average
  - RMSProp in practice
    - Effective and practical for deep neural networks
    - Popular

# RMSProp

**Require:** Global learning rate  $\epsilon$ , decay rate  $\rho$ .

**Require:** Initial parameter  $\theta$

**Require:** Small constant  $\delta$ , usually  $10^{-6}$ , used to stabilize division by small numbers.

Initialize accumulation variables  $r = 0$

**while** stopping criterion not met **do**

- Sample a minibatch of  $m$  examples from the training set  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  with corresponding targets  $\mathbf{y}^{(i)}$ .
- Compute gradient:  $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$
- Accumulate squared gradient:  $\mathbf{r} \leftarrow \rho \mathbf{r} + (1 - \rho) \mathbf{g} \odot \mathbf{g}$
- Compute parameter update:  $\Delta \theta = -\frac{\epsilon}{\sqrt{\delta+r}} \odot \mathbf{g}$ . ( $\frac{1}{\sqrt{\delta+r}}$  applied element-wise)
- Apply update:  $\theta \leftarrow \theta + \Delta \theta$

**end while**

[Goodfellow et al.]

11

# RMSProp with Nesterov Momentum

**Require:** Global learning rate  $\epsilon$ , decay rate  $\rho$ , momentum coefficient  $\alpha$ .

**Require:** Initial parameter  $\theta$ , initial velocity  $v$ .

Initialize accumulation variable  $r = 0$

**while** stopping criterion not met **do**

- Sample a minibatch of  $m$  examples from the training set  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  with corresponding targets  $\mathbf{y}^{(i)}$ .
- Compute interim update:  $\tilde{\theta} \leftarrow \theta + \alpha v$
- Compute gradient:  $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\theta}} \sum_i L(f(\mathbf{x}^{(i)}; \tilde{\theta}), \mathbf{y}^{(i)})$
- Accumulate gradient:  $\mathbf{r} \leftarrow \rho \mathbf{r} + (1 - \rho) \mathbf{g} \odot \mathbf{g}$
- Compute velocity update:  $v \leftarrow \alpha v - \frac{\epsilon}{\sqrt{r}} \odot \mathbf{g}$ . ( $\frac{1}{\sqrt{r}}$  applied element-wise)
- Apply update:  $\theta \leftarrow \theta + v$

**end while**

[Goodfellow et al.]

12

# Adam

- Adam (Kingma and Ba, 2014) — adaptive moments
- Combination of RMSProp and momentum, with following distinctive features
  - Momentum incorporated directly as an estimate of the first order moment (with exponential weighting) of the gradient
    - ♦ Apply momentum to rescaled gradients
    - This does not have clear theoretical justification
  - Bias corrections to estimates of
    - ♦ First-order moments (the momentum term)
    - ♦ Second-order (uncentered) moments to account for their initialization at the origin
  - RMSProp also includes estimate of uncentered second-order moment, but without correction
    - ♦ Therefore, in early stages of training, RMSProp second-order moment estimate may have high bias
- Adam in practice
  - Robust to choice of hyperparameters
  - Beware of suggested learning-rate default values — may need to change

J. Braun

13

# Adam

Require: Step size  $\epsilon$  (Suggested default: 0.001)  
Require: Exponential decay rates for moment estimates,  $\rho_1$  and  $\rho_2$  in  $[0, 1]$ .  
(Suggested defaults: 0.9 and 0.999 respectively)  
Require: Small constant  $\delta$  used for numerical stabilization. (Suggested default:  
 $10^{-8}$ )  
Require: Initial parameters  $\theta$   
Initialize 1st and 2nd moment variables  $s = 0, r = 0$   
Initialize time step  $t = 0$   
while stopping criterion not met do  
    Sample a minibatch of  $m$  examples from the training set  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  with  
    corresponding targets  $\mathbf{y}^{(i)}$ .  
    Compute gradient:  $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$   
     $t \leftarrow t + 1$   
    Update biased first moment estimate:  $\hat{s} \leftarrow \rho_1 s + (1 - \rho_1) \mathbf{g}$   
    Update biased second moment estimate:  $\hat{r} \leftarrow \rho_2 r + (1 - \rho_2) \mathbf{g} \odot \mathbf{g}$   
    Correct bias in first moment:  $\hat{s} \leftarrow \frac{\hat{s}}{1 - \rho_1^t}$   
    Correct bias in second moment:  $\hat{r} \leftarrow \frac{\hat{r}}{1 - \rho_2^t}$   
    Compute update:  $\Delta\theta = -\epsilon \frac{\hat{s}}{\sqrt{\hat{r} + \delta}}$  (operations applied element-wise)  
    Apply update:  $\theta \leftarrow \theta + \Delta\theta$   
end while

Exponential  
moving  
average

14

# **Questions?**

---

# **Neural Networks and Deep Learning**

**Dr. Jerome J. Braun**

## **This Lecture: TensorFlow Essentials II**

Course: Neural Networks and Deep Learning  
IE 7615

---

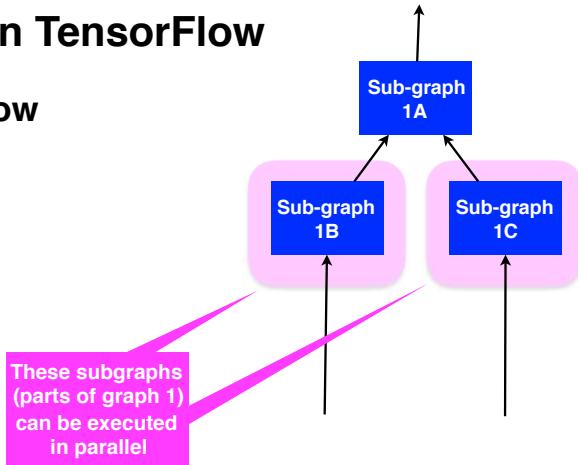
J. Braun

---

**Unauthorized distributing, redistributing, posting,  
and/or reposting, of the materials of this course  
(including course lectures, lecture slides, slide-sets,  
syllabi, guidelines, assignments, quizzes, exams,  
presentations, software, electronic media, etc.),  
is prohibited.**

# Recap: TensorFlow Basics (cont.)

- Recall computational graphs
  - Introduced in previous lecture
- Computation graphs in TensorFlow
  - Key feature in TensorFlow



## tf.GradientTape API (TF v2)

You should use the most-recent [STABLE](#) version of Release 2 (v2.11.0 or later stable version)

- API for automatic differentiation
- Compute gradient of computation w.r.t. input variables
  - Operations inside `tf.GradientTape` “recorded” to “tape”
  - Tape and gradients (associated with each recorded operation)
    - ♦ Used subsequently to compute gradients of “recorded” computation
    - By reverse-mode differentiation

# tf.GradientTape API

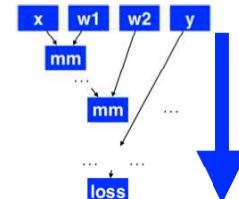
```
x = tf.convert_to_tensor ....  
y = tf.convert_to_tensor ....  
w1 = tf.Variable(tf.random.uniform ....  
w2 = tf.Variable(tf.random.uniform ....
```

Could use other names, e.g., "as t")

with `tf.GradientTape()` as `tape`

"Record to tape"

```
h = tf.maximum(tf.matmul(x, w1), 0)  
y_pred = tf.matmul(h, w2)  
diff = y_pred - y  
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))
```



...

**Forward pass**

All operations within GradientTape context traced (recorded) for computing gradients later

J. Braun

5

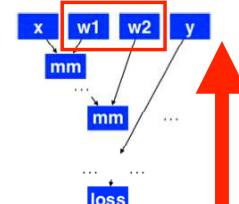
# tf.GradientTape API

```
x = tf.convert_to_tensor ....  
y = tf.convert_to_tensor ....  
w1 = tf.Variable(tf.random.uniform ....  
w2 = tf.Variable(tf.random.uniform ....
```

with `tf.GradientTape()` as `tape`

```
h = tf.maximum(tf.matmul(x, w1), 0)  
y_pred = tf.matmul(h, w2)  
diff = y_pred - y  
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))
```

```
gradients = tape.gradient(loss, [w1, w2])
```



**Backward pass**

`tape.gradient` uses traced computation-graph to compute gradients

J. Braun

6

# TensorFlow Datasets – TFDS (Recap)

TF 2.11.0

```
import tensorflow as tf  
import tensorflow_datasets as tfds
```

`tf.__version__` → '2.11.0'

`tfds.__version__` → '4.8.2'

`dataset = tfds.load(name='.....')`

Dataset name  
• See names of available datasets on tensorflow DOT org

→ Downloading and preparing dataset

...

J. Braun

7

# TensorFlow Datasets – TFDS (cont.)

```
import tensorflow as tf  
import tensorflow_datasets as tfds
```

You should use the most-recent STABLE version of Release 2 (v2.11.0 or later stable version)

`print(tfds.list_builders())` → List available datasets

['abstract\_reasoning', ..., 'cifar10', 'cifar100', ...,  
'cnn\_dailymail', 'coco', ..., 'fashion\_mnist', ...,  
'imagenet\_v2', ..., 'yelp\_polarity\_reviews', ..., 'yes\_no']

`dataset = tfds.load(name="mnist", split="train")` → Construct tf.data.Dataset

Build input pipeline

```
dataset =  
dataset.shuffle(1024).batch(32).prefetch(tf.data.experimental.AUTOTUNE)
```

```
for features in dataset.take(1):  
    image, label = features["image"], features["label"]
```

J. Braun

8

# Questions?

---

---

# **Neural Networks and Deep Learning**

**Dr. Jerome J. Braun**

## **This Lecture: Convolutional Neural Networks I**

Course: Neural Networks and Deep Learning  
IE 7615

---

J. Braun

---

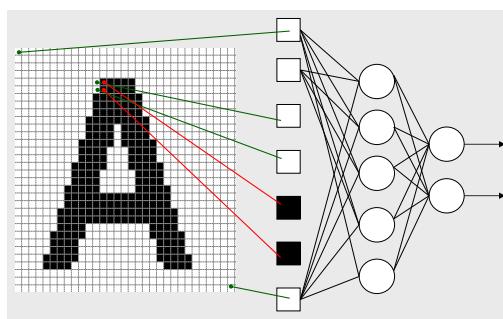
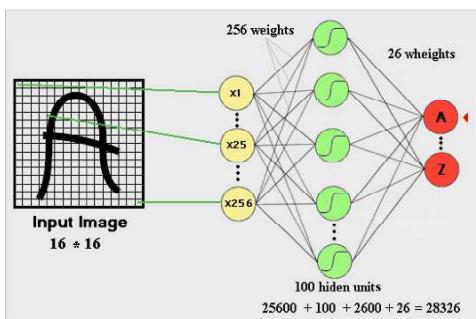
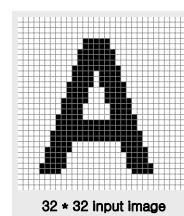
**Unauthorized distributing, redistributing, posting,  
and/or reposting, of the materials of this course  
(including course lectures, lecture slides, slide-sets,  
syllabi, guidelines, assignments, quizzes, exams,  
presentations, software, electronic media, etc.),  
is prohibited.**

# This Lecture

- · Application motivation – character recognition
- Features and representation learning
- CNN as deep network architectures
- CNN inspiration roots (neurobiological)
- CNN early idea
- Convolution operation definition

## Motivation for CNNs (Character Recognition)

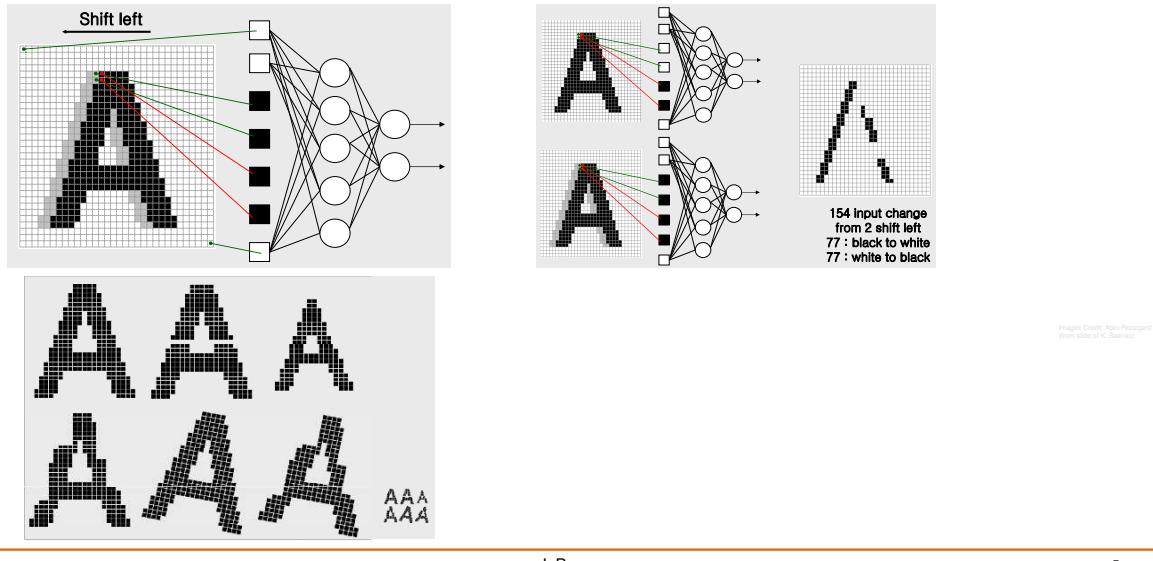
- “Naive” utilization of MLP ANN (inappropriate)
  - Large number of input parameters
  - Large number of weights



## Motivation for CNNs (Character Recognition)

- Inappropriate utilization of MLP ANNs

- Non-robust to shifts, scaling, distortions



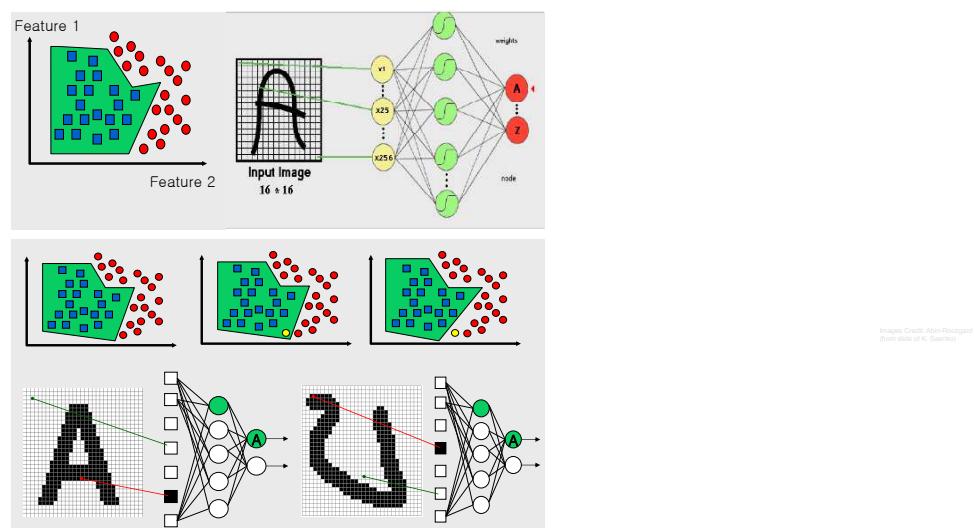
J. Braun

5

## Motivation for CNNs (Character Recognition)

- Inappropriate utilization of MLP ANNs

- Ignores structure in data



J. Braun

6

# Character Recognition with MLP FNN

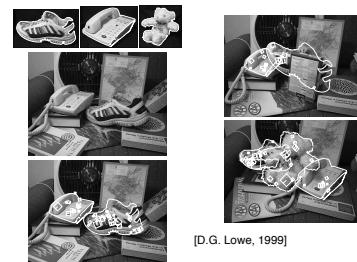
- **Sufficiently (fully) connected MLP FNN network of sufficient size**
  - May achieve resilience to input variations (invariance)
  - Issues
    - ♦ Network size
    - ♦ Training time
    - ♦ Training performance
    - ♦ Free parameters
- **Conventional alternative — “clever” design of features**

J. Braun

7

# SIFT Features

- **SIFT — Scale-Invariant Feature Transform (Lowe, 1999, Lowe 2004)**
  - Example of human-engineered (“hand-crafted”) features to achieve invariance
- **SIFT features for image**
  - Invariant to
    - ♦ Scaling
    - ♦ Translation
    - ♦ Rotation
  - Partially invariant to
    - ♦ Changes of illumination
    - ♦ Affine or 3D projection



[D.G. Lowe, 1999]

J. Braun

8

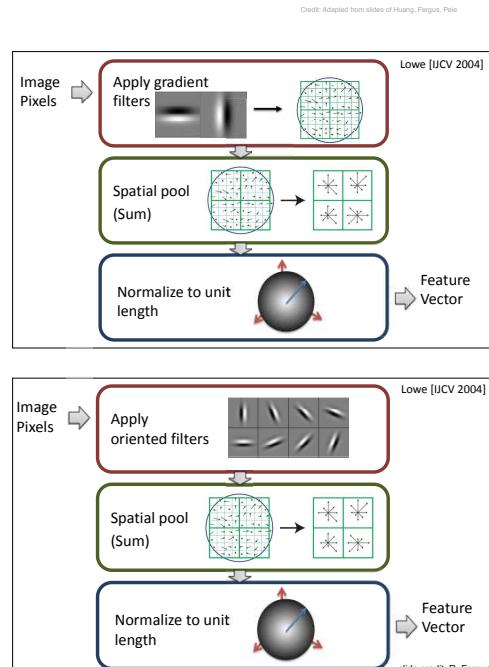
# SIFT Overview

## • Detector

- Find scale-space extrema
- Keypoint localization and filtering
  - Improve key points
  - Discard bad key points

## • Descriptor

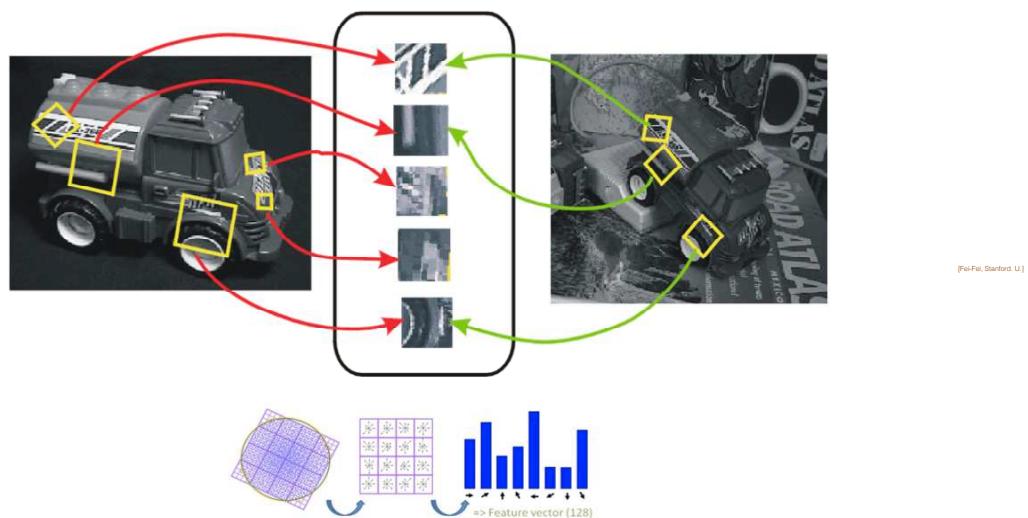
- Orientation assignment
  - Remove effects of rotation and scale
- Create descriptor
  - Using histograms of orientation



9

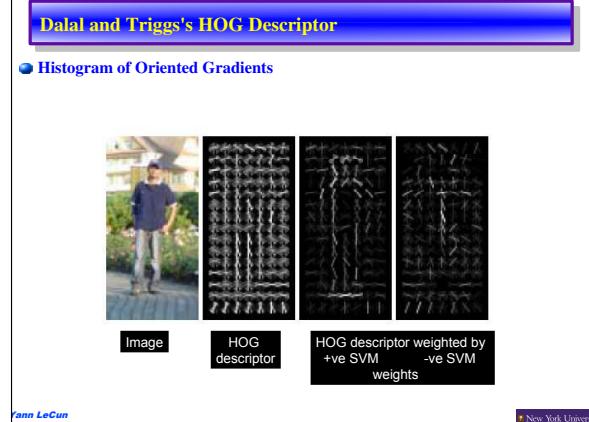
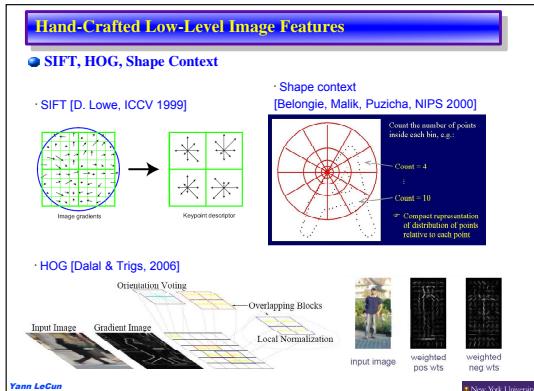
# SIFT

## • SIFT for object recognition (Lowe, 1999)



10

## “Hand-Crafted” Invariance to Shifts and Rotations



J. Braun

11

## This Lecture

- Application motivation – character recognition
- Features and representation learning
- • CNN as deep network architectures
- CNN inspiration roots (neurobiological)
- CNN early idea
- Convolution operation definition

J. Braun

12

# CNNs as Deep Networks

---

- Use raw input data directly
  - Dispense with feature-extraction stage
    - ♦ No “feature engineering”
    - No human-designed (hand-crafted) features
- Deep Learning for image processing
  - Low-level features → mid-level features → object IDs
  - Multiple layers
    - Representations increasingly invariant

# CNNs as Deep Networks (cont.)

---

- Recall from previous lectures:
  - Loss-surface in deep networks
    - Dimensionality
    - Non-convex
    - Ill-conditioned
    - Saddle-points, flat regions
- CNNs — deep network architectures
  - Y. LeCun (c. 2010):
    - *“For reasons that are not well understood theoretically, back-prop works well when they are highly structured, e.g., convolutional networks”*

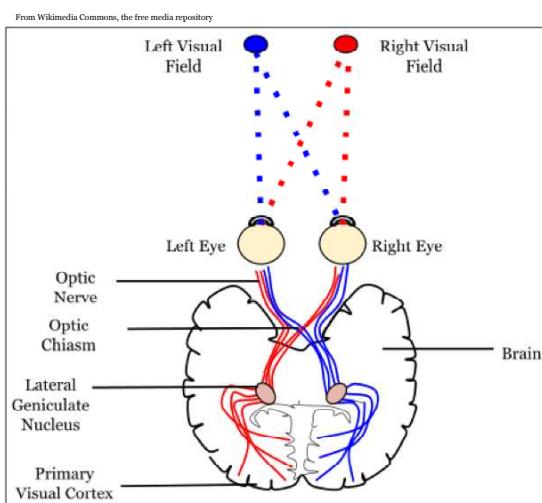
# This Lecture

- Application motivation – character recognition
- Features and representation learning
- CNN as deep network architectures
- **CNN inspiration roots (neurobiological)** →
- CNN early idea
- Convolution operation definition

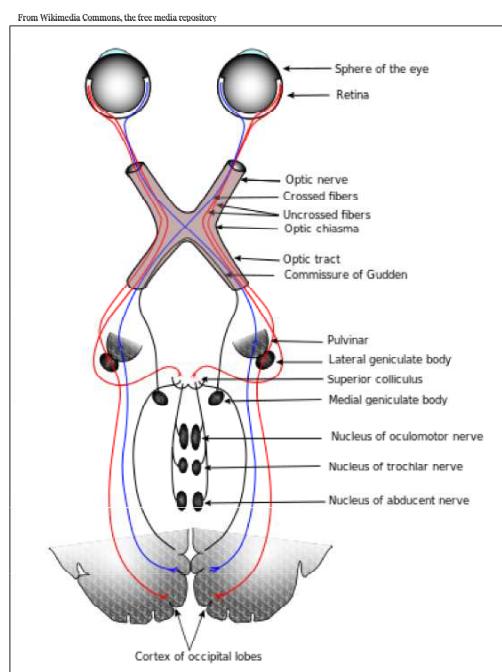
J. Braun

15

# Human Vision



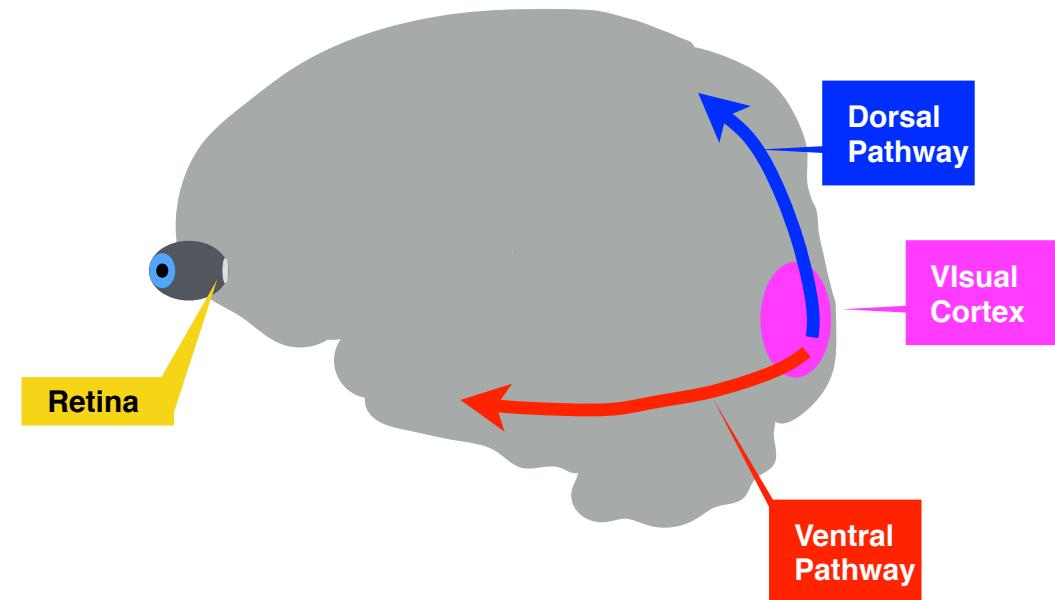
- Visual Cortex
- Caudal (posterior) view
- Brodmann areas 17, 18, 19



J. Braun

16

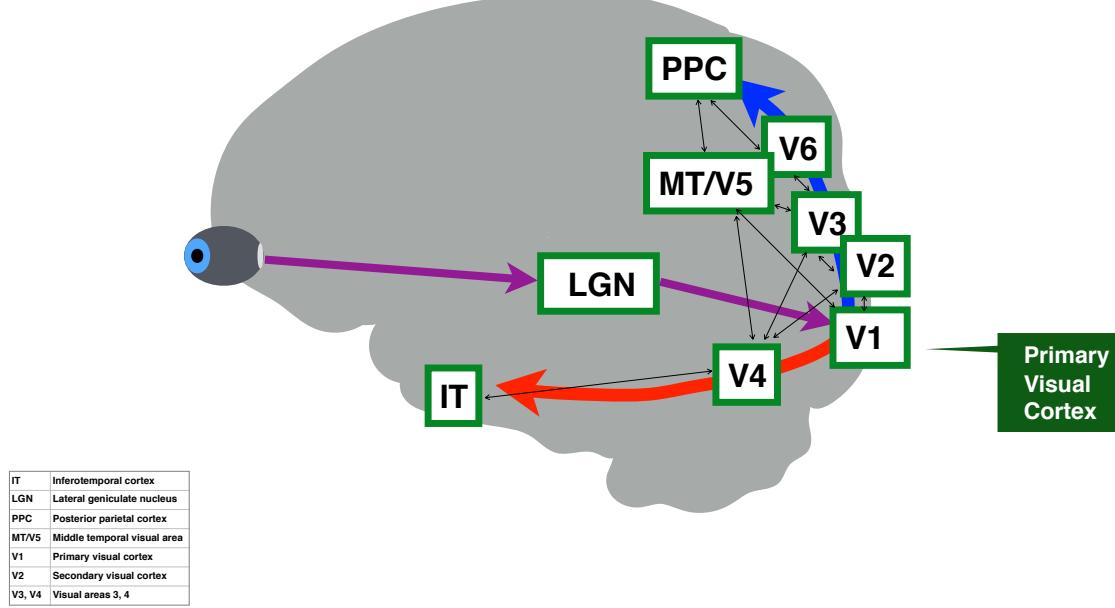
## Pathways From Primary Visual Cortex



J. Braun

17

## Visual Information Flow (Simplified)



J. Braun

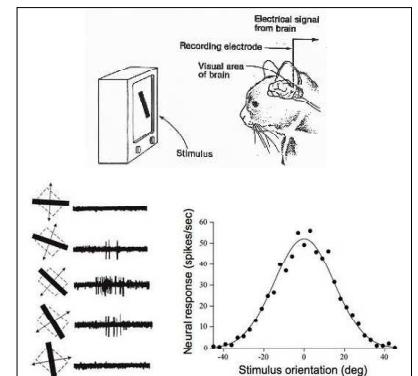
18

# D. Hubel and T. Wiesel

- 1959
  - Receptive fields of single neurons in cat's striate cortex
- 1962
  - Receptive fields, binocular interaction, functional architecture in visual cortex (cat)
- Locally-sensitive, orientation-selective cells in visual cortex (cat)
- Receptive fields
- Simple cells detect local features
- Complex cells pool outputs within retinotopic neighborhood



Image credit: From slide of Jia-Bin Huang, U. of Illinois



[Hubel & Wiesel]

Images credit: From slide of Fei-Fei Li, A. Karpathy, J. Johnson, Stanford U.

J. Braun

19

## Neocognitron (Fukushima, Biological Cybernetics 1980)

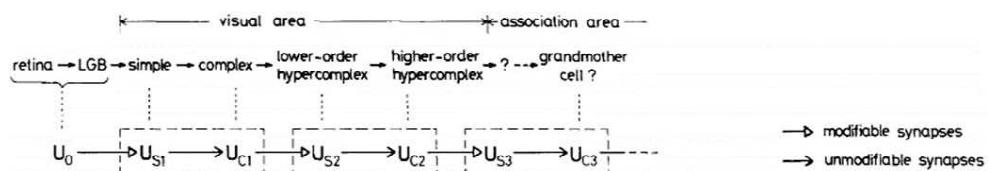
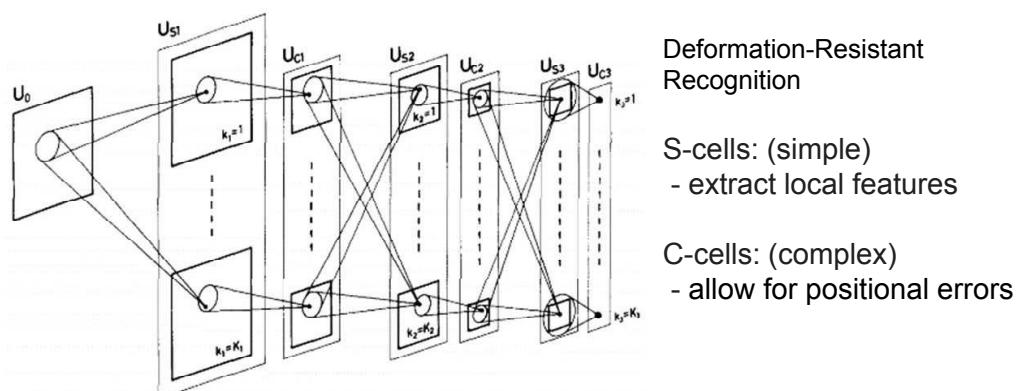


Fig. 1. Correspondence between the hierarchy model by Hubel and Wiesel, and the neural network of the neocognitron



Deformation-Resistant  
Recognition

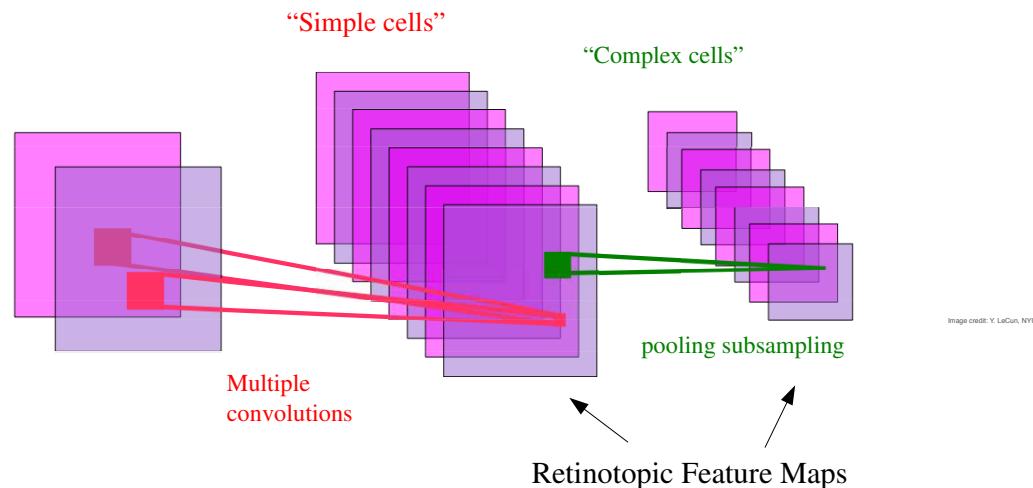
S-cells: (simple)  
- extract local features

C-cells: (complex)  
- allow for positional errors

20

# Convolutional Neural Network — Early Idea

- Yan LeCun — Convolutional Neural Networks
  - Inspired by findings of Hubel and Wiesel



# Convolution Operation

- Consider 1D signal  $x(t)$  and denoising problem
- Denoise by averaging noisy signal
  - Weigh raw signal
  - Samples close to current signal-value  $x(t)$  matter more
- If done for all points  $t$ :  $s(t) = \int x(a)w(t-a)da$ 
  - Convolution operation
  - Often denoted as:  $s(t) = (x * w)(t)$

## Convolution Operation (cont.)

$$s(t) = (x * w)(t)$$

$$s(t) = \int x(a)w(t-a)da$$

|      \backslash

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n)$$

Feature map

Input

Kernel  
(Filter)

CNN  
Terminology

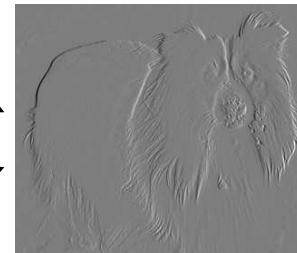
J. Braun

23

## Edge Detection by Convolution



Input



Output

1	-1
---	----

Kernel

24

# Convolution and Pooling Layers

---

- **Convolution layers**
  - Weight sharing
- **Pooling/subsampling layers**
  - Invariance to small distortions
- **Back-prop based supervised gradient descent**
  - All layers trained simultaneously

Will discuss these in detail in next lectures

## Questions?

---

---

# **Neural Networks and Deep Learning**

**Dr. Jerome J. Braun**

## **This Lecture: Convolutional Neural Networks II**

Course: Neural Networks and Deep Learning  
IE 7615

---

J. Braun

---

**Unauthorized distributing, redistributing, posting,  
and/or reposting, of the materials of this course  
(including course lectures, lecture slides, slide-sets,  
syllabi, guidelines, assignments, quizzes, exams,  
presentations, software, electronic media, etc.),  
is prohibited.**

# This Lecture

---

- **Convolution**
- **Convolution layers**
- **Pooling layers**
- **CNN architecture basics**

## CNNs as Deep Networks (Recap)

---

- **Use raw input data directly**
  - Dispense with feature-extraction stage
    - ♦ No “feature engineering”
      - No human-designed (hand-crafted) features
- **Deep Learning for image processing**
  - Low-level features → mid-level features → object IDs
  - Multiple layers
    - Representations increasingly invariant

# CNNs as Deep Networks (cont.) (Recap)

- Recall from previous lectures:
  - Loss-surface in deep networks
    - Dimensionality
    - Non-convex
    - Ill-conditioned
    - Saddle-points, flat regions
- CNNs — deep network architectures
  - Y. LeCun (c. 2010):
    - ◆ “*For reasons that are not well understood theoretically, back-prop works well when they are highly structured, e.g., convolutional networks*”

J. Braun

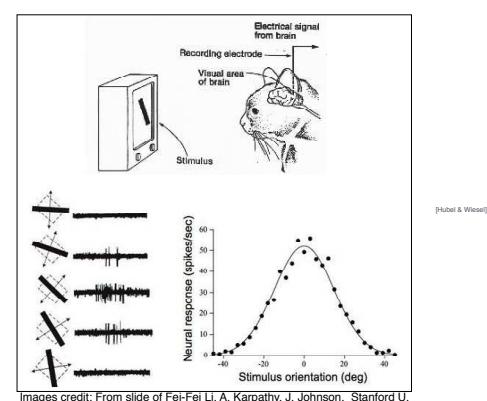
5

# D. Hubel and T. Wiesel (Recap)

- 1959
  - Receptive fields of single neurons in cat's striate cortex
- 1962
  - Receptive fields, binocular interaction, functional architecture in visual cortex (cat)
- Locally-sensitive, orientation-selective cells in visual cortex (cat)
- Receptive fields
- Simple cells detect local features
- Complex cells pool outputs within retinotopic neighborhood



Image credit: From slide of Jia-Bin Huang, U. of Illinois



Images credit: From slide of Fei-Fei Li, A. Karpathy, J. Johnson, Stanford U.

J. Braun

6

## Convolution and Pooling Layers (Recap)

---

- **Convolution layers**
  - Weight sharing
- **Pooling/subsampling layers**
  - Invariance to small distortions
- **Back-prop based supervised gradient descent**
  - All layers trained simultaneously

## Convolution Operation (Recap)

---

- Consider 1D signal  $x(t)$  and denoising problem
- Denoise by averaging noisy signal
  - Weigh raw signal
  - Samples close to current signal-value  $x(t)$  matter more
- If done for all points  $t$ :  $s(t) = \int x(a)w(t-a)da$ 
  - Convolution operation
  - Often denoted as:  $s(t) = (x * w)(t)$

## Convolution Operation (cont.) (Recap)

$$s(t) = (x * w)(t)$$

$$s(t) = \int x(a)w(t-a)da$$

|      \backslash

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n)$$

Feature map

Input

Kernel  
(Filter)

CNN  
Terminology

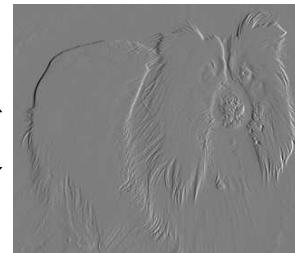
J. Braun

9

## Edge Detection by Convolution (Recap)



Input



Output

1	-1
---	----

Kernel

10

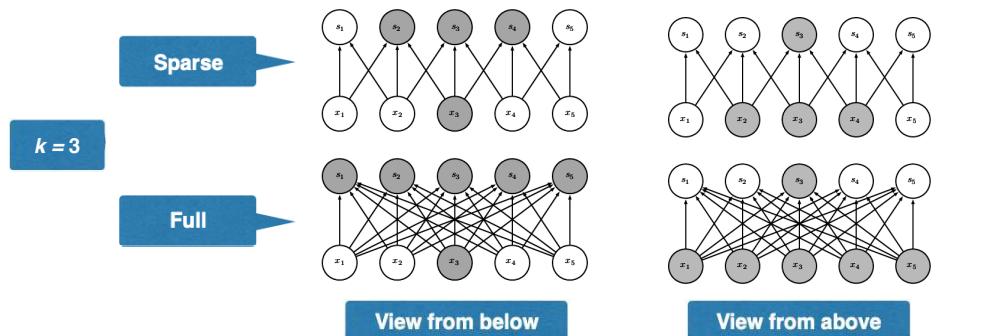
# Role of Convolution in Neural Network

- Sparse interactions
- Parameter sharing
- Equivariant representations
- Mechanism for variable-size inputs

Explained in  
following  
slides

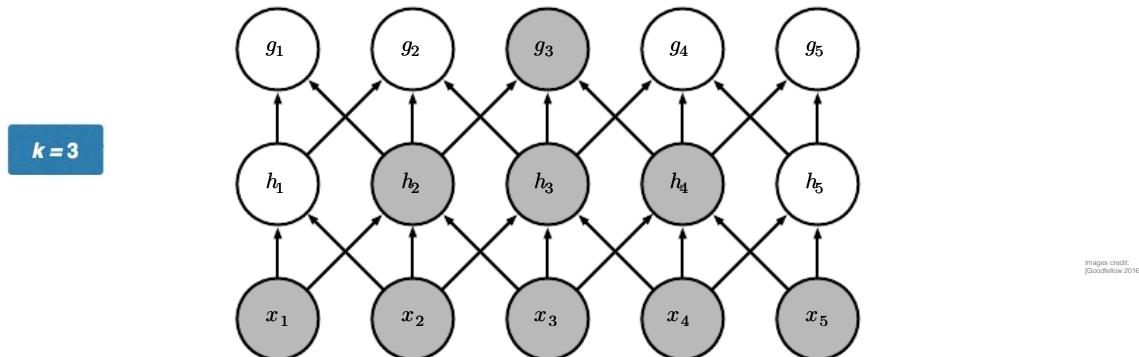
## Sparse Interactions

- “Traditional” neural network full connectivity:  $w^T x$ 
  - Matrix multiplication —  $m \times n$  parameters,  $O(m \times n)$  complexity (per example)
- CNN: size of kernel smaller than size of input
  - Sparse connectivity, sparse weights — store fewer parameters
    - Reduces memory requirements
    - Improves statistical efficiency of model
    - Fewer operations to compute output
  - Suppose:  $m$  inputs,  $n$  outputs where  $n < m$ 
    - $k$  — number of connections for each output
    - $k \times n$  parameters
    - Run-time complexity:  $O(k \times n)$
    - In practice, good performance possible with  $k \ll m$  by several orders of magnitude



# Sparse Interactions and CNN Depth

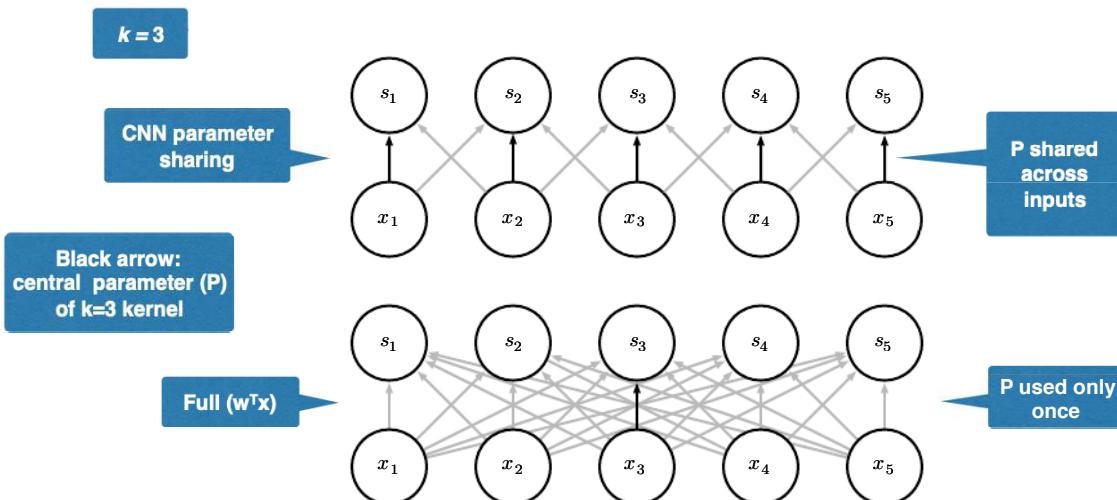
- Receptive field in deeper layers larger than in shallow layers
  - Architectural features, e.g, stride and pooling, compound this (will see shortly)
- Therefore:
  - Although *direct* connectivity sparse, deeper layers connected to all or most of inputs



J. Braun

13

# Parameter Sharing in CNN



J. Braun

14

# Convolution Layer

Example: 32x32x3 image

Height: 32

Width: 32

Depth: 3

J. Braun

15

# Convolution Layer

Example: 32x32x3 image

Height: 32

Example:  
5x5x3 kernel (filter)



Depth 3

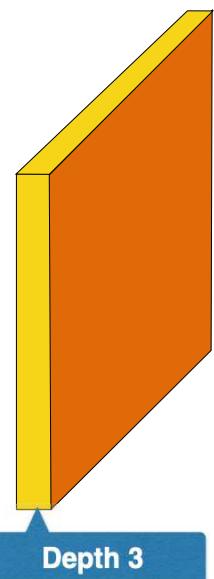
Width: 32

Depth: 3

J. Braun

16

# Convolution



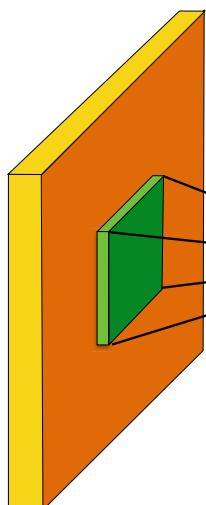
Example:  
5x5x3 kernel (filter)



Depth 3

- Convolution in 2D
- By analogy to 1D signal convolution
- Kernel (filter) convolved with image by spatial products
  - In height and width
  - Must have same depth

# Convolution



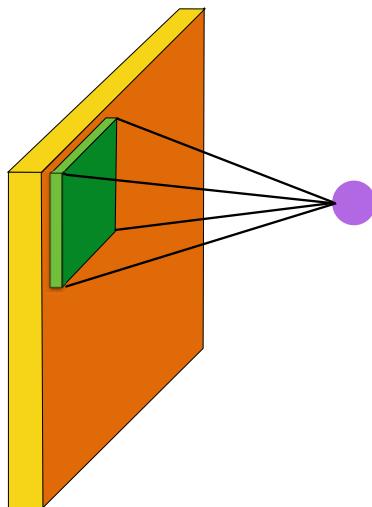
$$w^T x + w_0$$

- Convolve kernel  $w$  with image parts  $x$ 
  - Part size = filter size, e.g., 5x5
- Add bias
  - e.g., 5x5x3 dot product plus bias

# Convolution

---

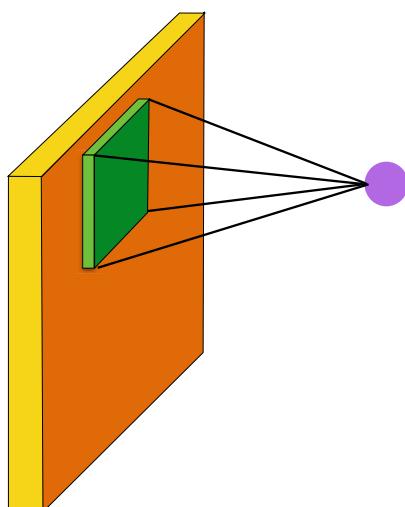
- Now slide kernel over all parts of image



# Convolution

---

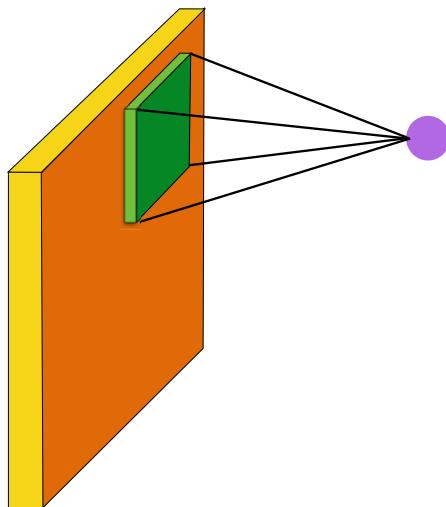
- Now slide kernel over all parts of image



# Convolution

---

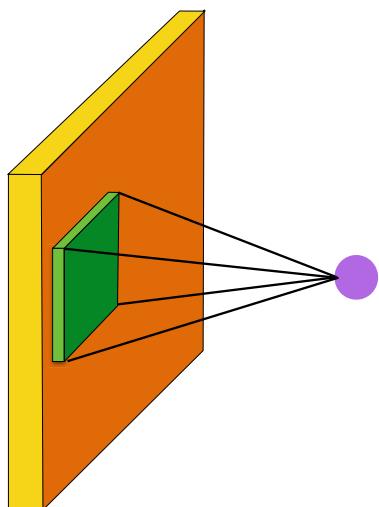
- Now slide kernel over all parts of image



# Convolution

---

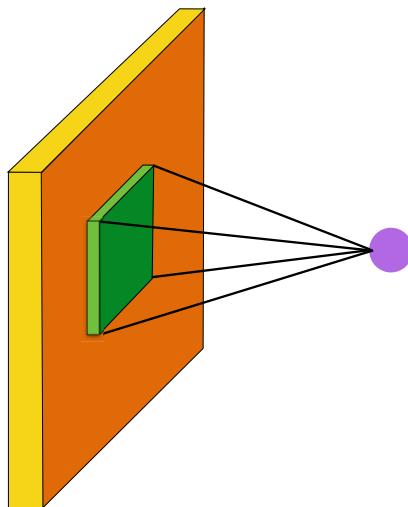
- Now slide kernel over all parts of image



# Convolution

---

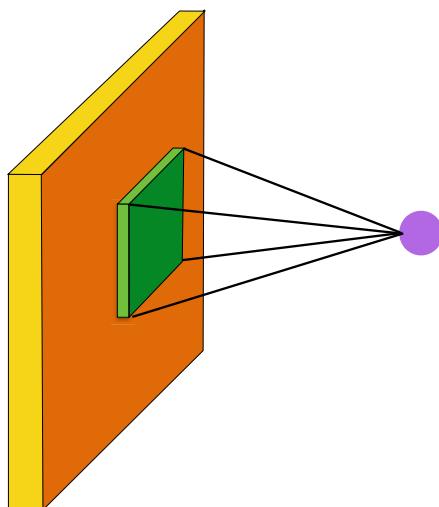
- Now slide kernel over all parts of image



# Convolution

---

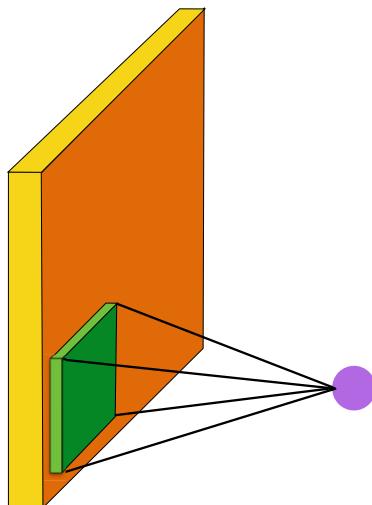
- Now slide kernel over all parts of image



# Convolution

---

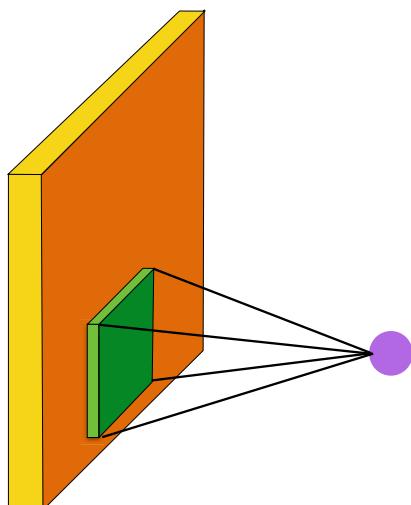
- Now slide kernel over all parts of image



# Convolution

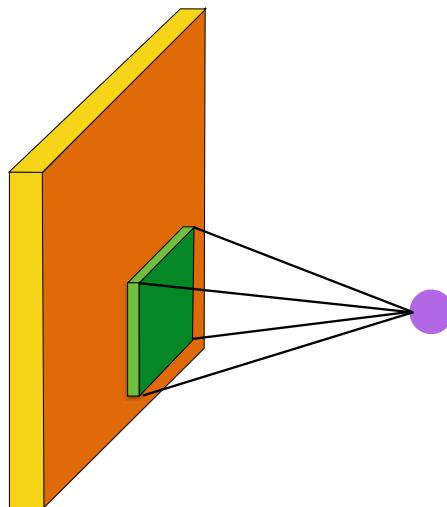
---

- Now slide kernel over all parts of image



# Convolution

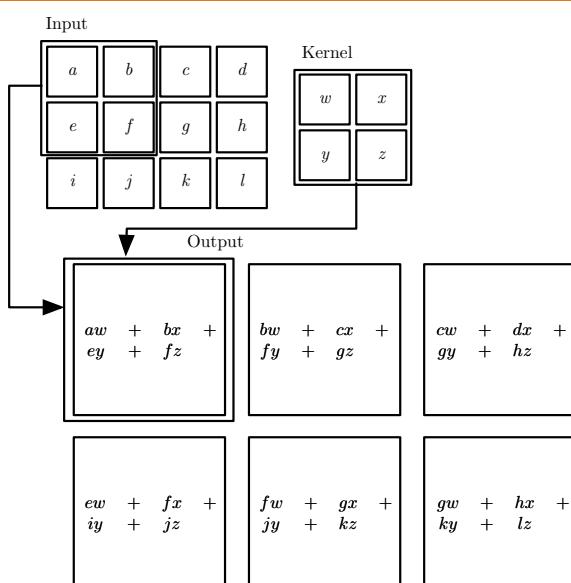
- Now slide kernel over all parts of image



J. Braun

27

# 2D Convolution

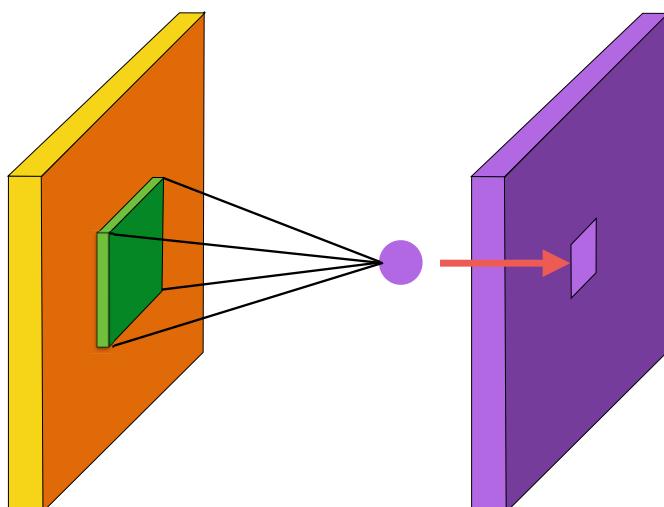


[Goodfellow 2016]

28

# Activation Map

- 2D structure
- As many elements as number of “fits” of kernel into image
- For 32x32x3 image and 5x5x3 kernel:
  - 28x28x1

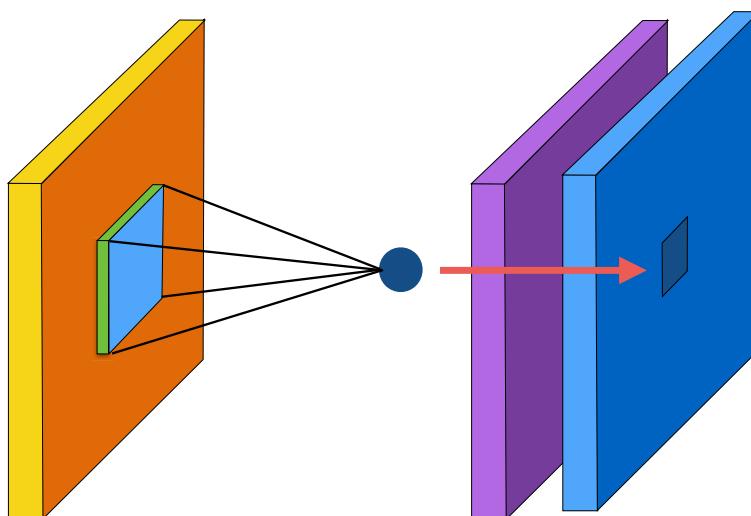


J. Braun

29

# Multiple Kernels

- Another kernel with different weights

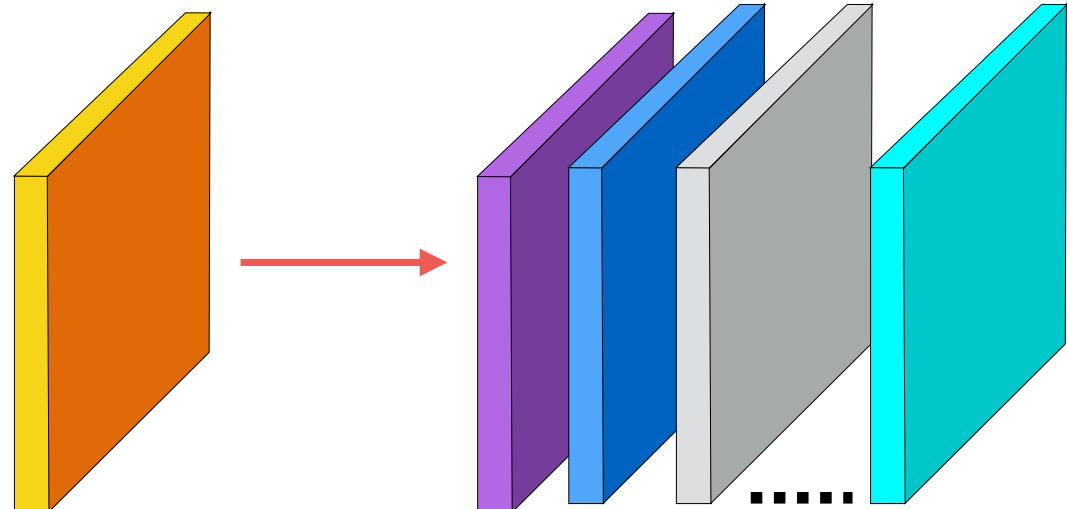


J. Braun

30

# Multiple Kernels

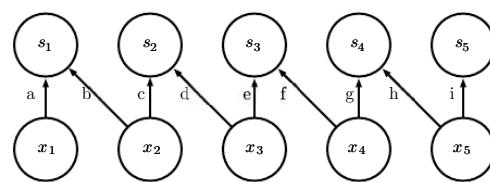
- Several activation maps



# Connectivity

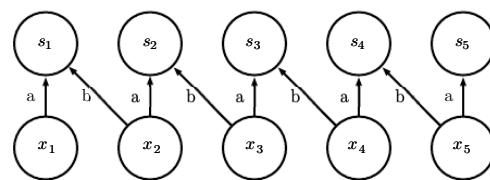
## Local connections

- Restricted connectivity
- No parameter sharing



## Convolution

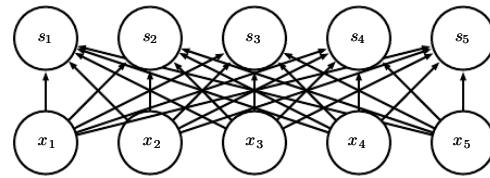
- Same connectivity restriction
- Parameter sharing



Images credit:  
[Goldfellow 2016]

## Fully connected

- No connectivity restriction
- No parameter sharing



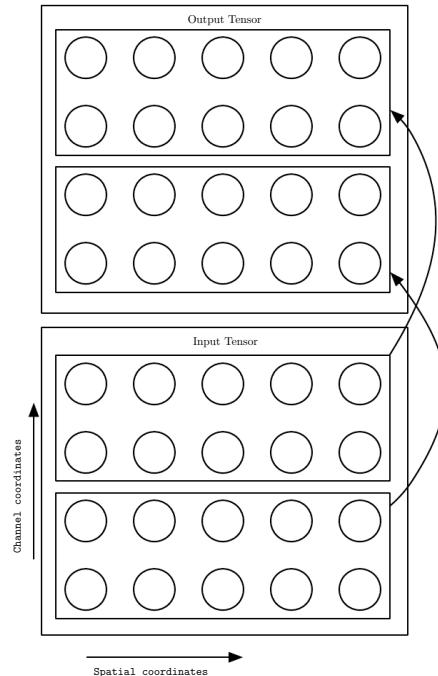
# Constraining Channel Connectivity

- Connectivity restricted further

- Each output channel constrained to be function of only *subset* of input channels
  - First  $m$  output channels connected to only first  $n$  input channels
  - Second  $m$  output channels connected to only second  $n$  input channels
  - etc.

- Restricting connectivity between channels lessens number of network parameters

- Benefits
  - Reduced memory consumption
  - Increased statistical efficiency
  - Reduced computation in forward- and back-propagation
- These benefits achieved *without* reducing number of hidden units



J. Braun

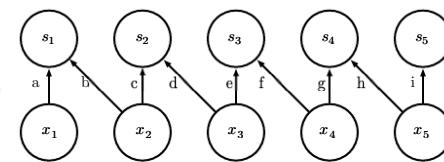
33

# Tiled Convolution

- Compromise between convolution layer and locally connected layer

Locally connected

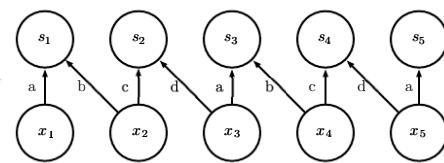
- a,b,c,d,e,f,g,h,i
- No sharing



- Cycle between kernels during convolution

Tiled, t=2

- Two kernels
- (a,b), (c,d)



- Different filters for *immediately-neighboring* locations

As in locally-connected layer

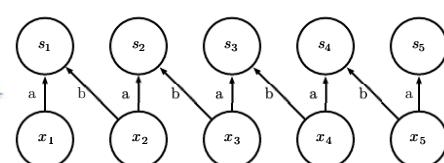
- Memory requirements for parameter storage

Increase only by factor of size of set of kernels

Not by size of entire output

Traditional convolution

- Equivalent to tiled with t=1

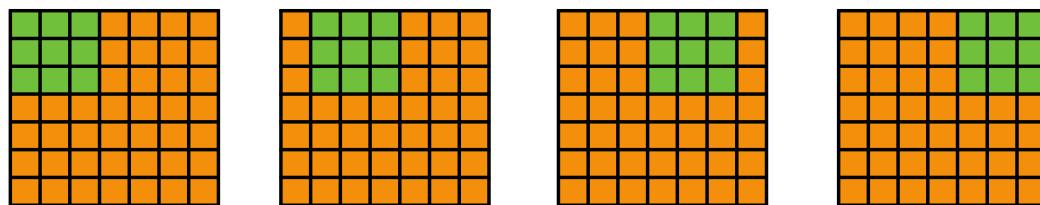
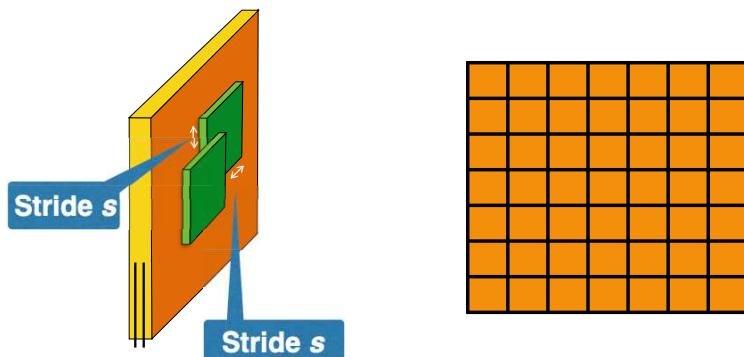


Images credit:  
[Goodfellow 2016]

J. Braun

34

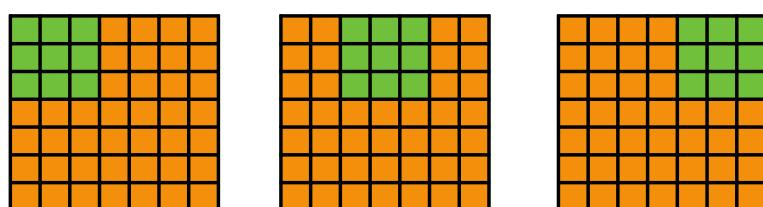
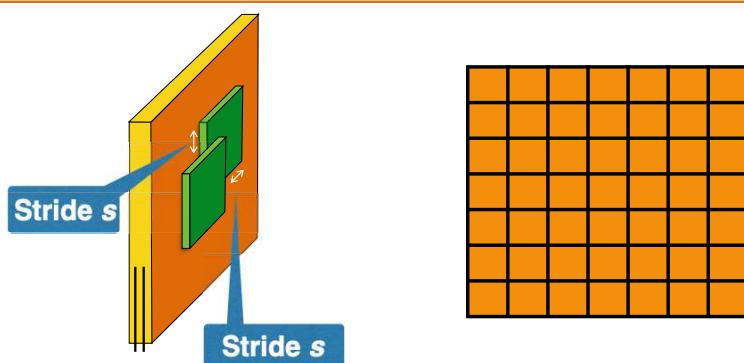
# Stride



J. Braun

35

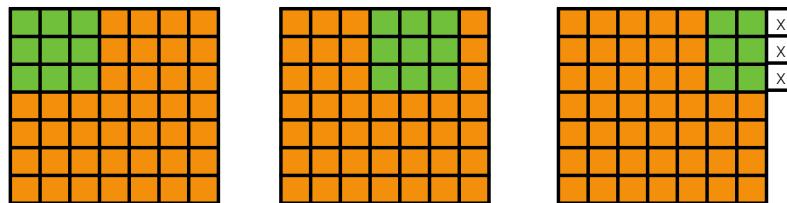
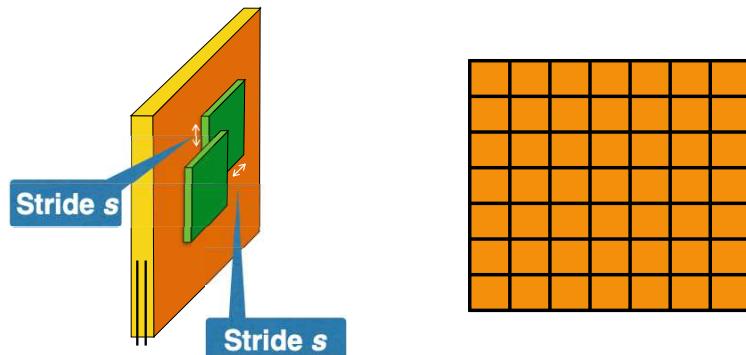
# Stride



J. Braun

36

# Stride

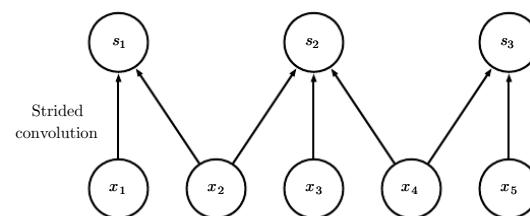


J. Braun

37

## Convolution with Stride

Convolution with stride  $s=2$

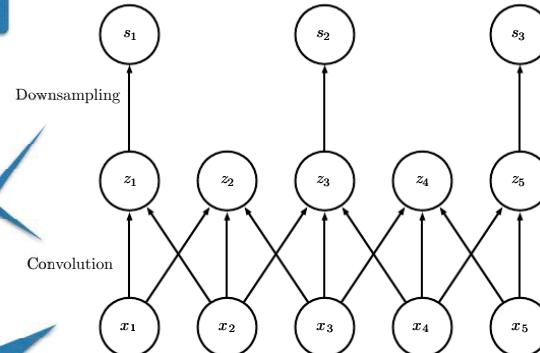


Computationally equivalent to:

Convolution with stride  $s=1$

Followed by downsampling

Computationally wasteful



J. Braun

38

# Pooling

- Downsample to make representations smaller
  - Efficiency
- Averaging or max pooling
  - Max pooling example
    - 2x2 kernel
    - Stride s=2

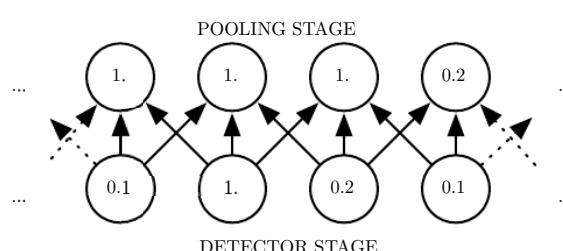
5	2	4	9
8	1	2	6
3	5	7	2
4	2	3	1

8	9
5	7

J. Braun

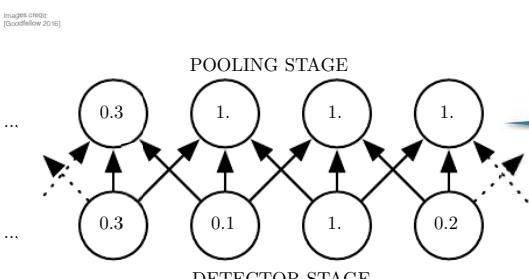
39

## Max Pooling and Invariance



**Pooling regions outputs**  
- Pooling region of 3 pixels  
- Stride s=1

“Detector stage” — nonlinear activation function, e.g., ReLu



After shift of input by one pixel

Only half of units changed

All units changed

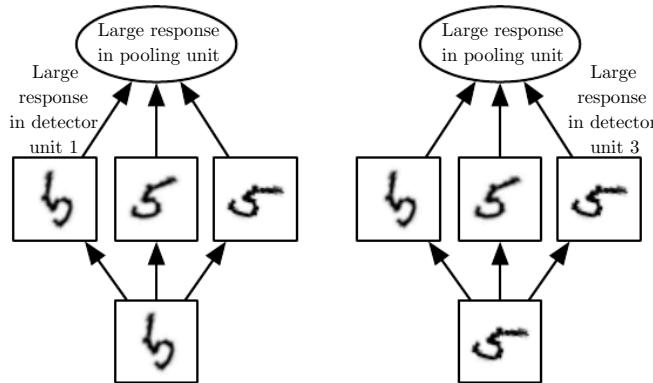
**Max pooling sensitive only to maximal value in neighborhood**  
- But not its exact location

J. Braun

40

## Pooling and Local Invariances – Example

- Pooling unit that pools over multiple features learned with separate parameters
  - Learns to be invariant to rotation (input digit “5”)
  - Three learned kernels and max pooling unit
  - Each kernel attempts to match slightly different orientation
  - Large activation in detector unit upon input = “5”
    - Effect on max pooling unit — roughly same



Max pooling over spatial positions invariant to translation

J. Braun

41

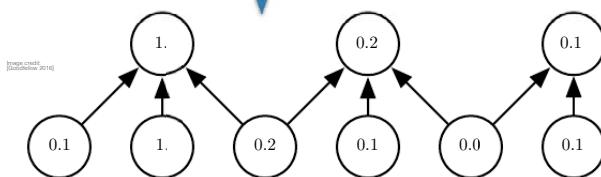
## Pooling with Downsampling

- Since pooling summarizes responses over entire neighborhood
  - Can use fewer pooling units than detector units
    - Report summary statistics for pooling regions spaced  $k$  pixels apart (rather than 1 apart)
- Reduces representation size
- Reduces computational and statistical burden on next layer

### Example max pooling

- Pool width of three
- Stride (between pools) of two

Next layer reduction



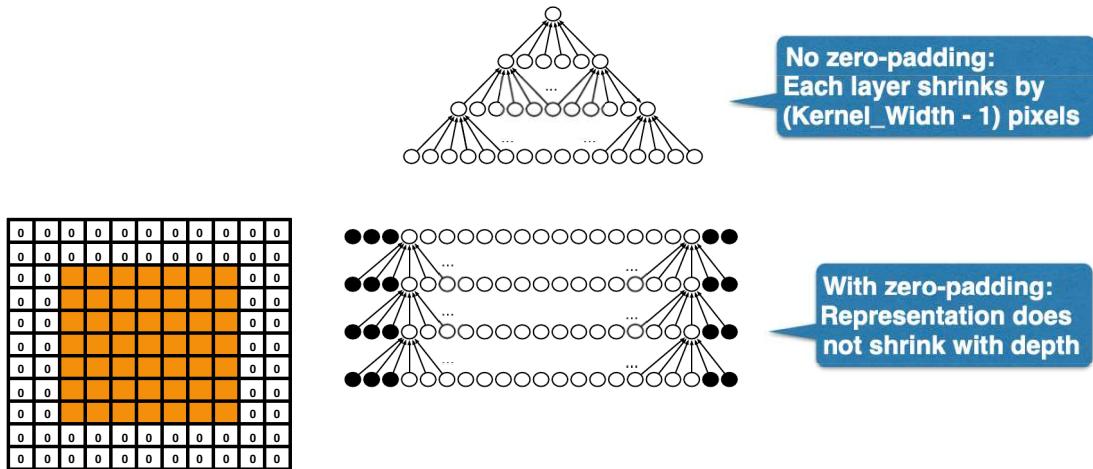
Rightmost pooling region has smaller size  
But must be included to avoid ignoring some of detector units

J. Braun

42

# Zero Padding

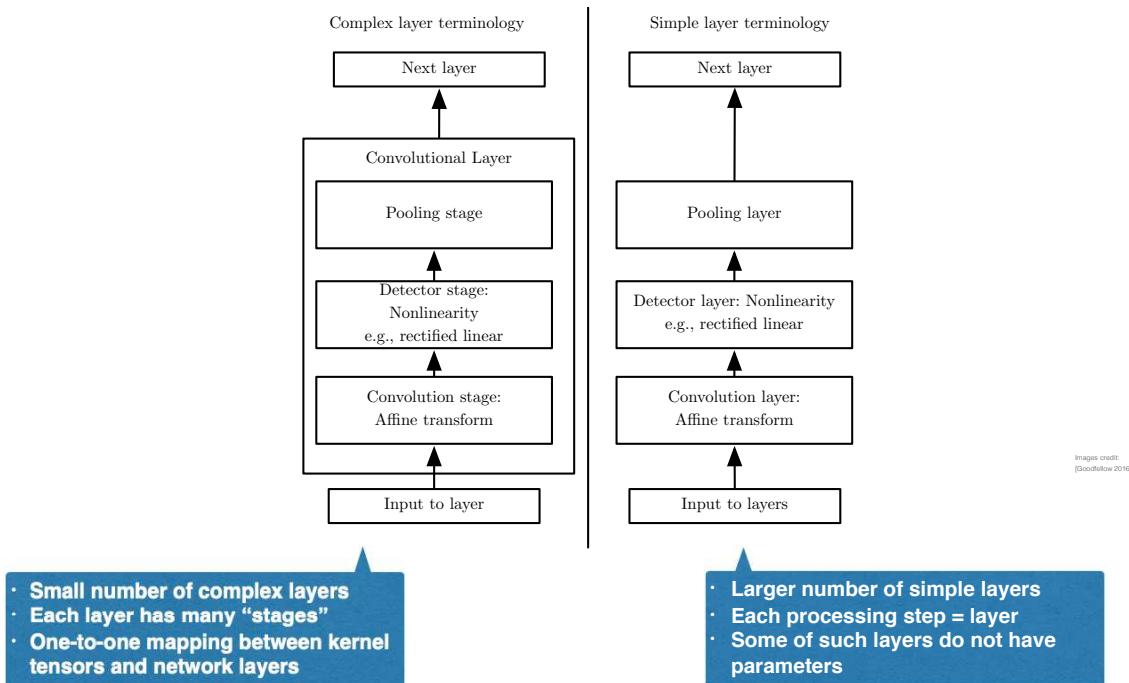
- Control layer size
- Prevent decrease of spatial dimensions in convolution operation



J. Braun

43

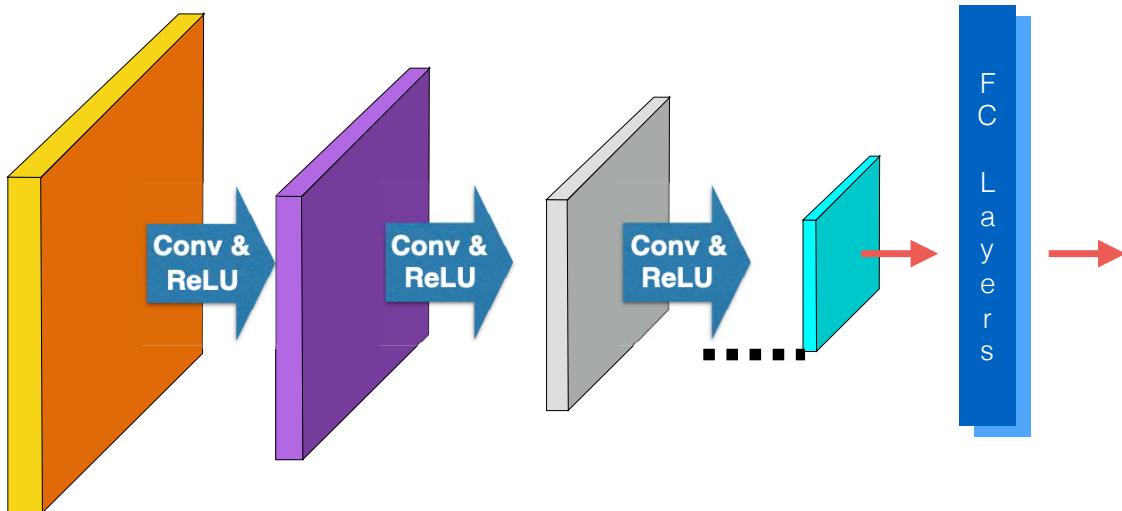
## CNN Layer Terminology Differences



J. Braun

44

# CNN — Simplified Architecture



- Progressively higher-level representations (features)

J. Braun

45

## Subsampling Layers

- Subsampling reduces spatial resolution of (each) feature map
  - Contributes to shift- and distortion-invariance

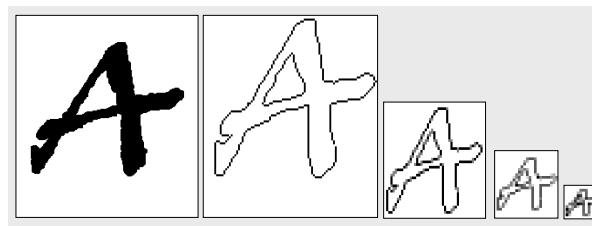
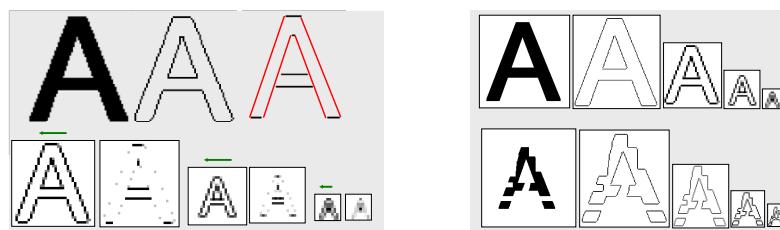


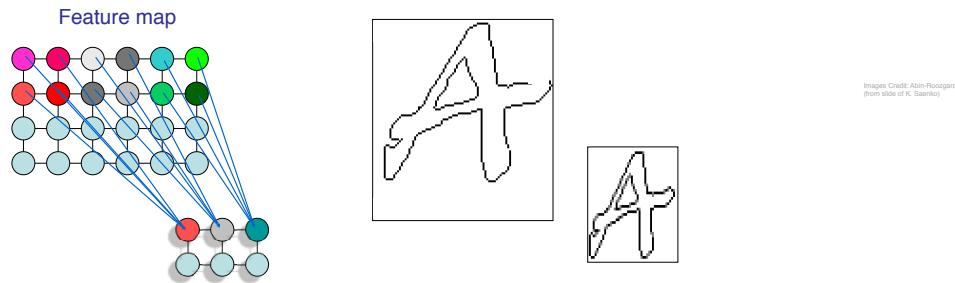
Image Credit: Alain-Rodrigard  
(from slide of K. Saenko)

J. Braun

46

# Subsampling Layers

- Subsampling reduces spatial resolution of (each) feature map
  - Contributes to shift- and distortion-invariance
- Weight-sharing applied to subsampling
  - Contributes to reducing noise and distortion

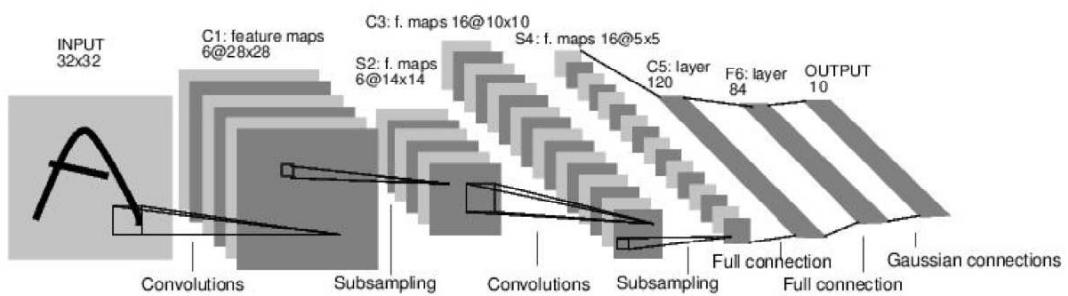


J. Braun

47

# Convolutional Neural Network – LeNet

- Yan Lecun (1980s)



[LeNet-5, LeCun 1980]

Image credit: From slide of Fei-Fei Li, A. Karpathy, J. Johnson, Stanford U.

J. Braun

48

## CNN For Handwriting Recognition (Handwritten Digits)

- Convolution layers – simple cells
  - Weight sharing
- Pooling/subsampling layers – complex cells
  - Invariance to small distortions
- Back-prop based supervised gradient descent
  - All layers trained simultaneously

(more on CNN BackProp shortly)

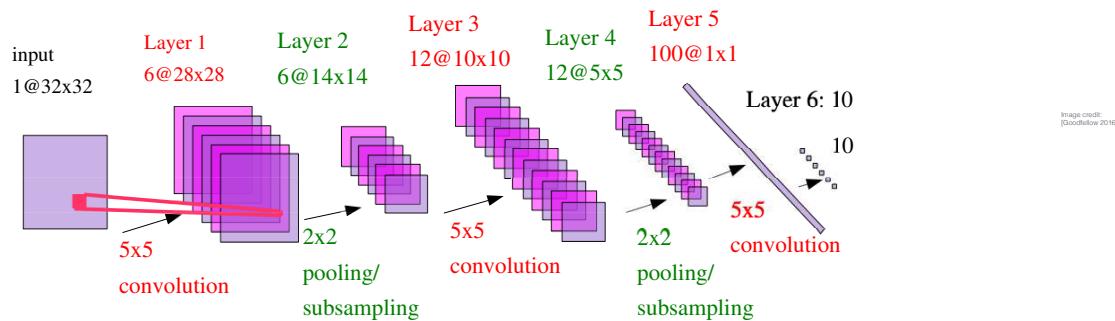


Image credit:  
(Goodfellow 2016)

J. Braun

49

### MNIST Handwritten Digit Dataset

3 6 8 1 7 9 6 6 4 1  
6 7 5 7 8 6 3 4 8 5  
2 1 7 9 7 1 2 8 4 6  
4 8 1 9 0 1 8 8 9 4  
7 6 1 8 6 4 1 5 6 0  
7 5 9 2 6 5 8 1 9 7  
2 2 2 2 2 3 4 4 8 0  
0 2 3 8 0 7 3 8 5 7  
0 1 4 6 4 6 0 2 4 3  
7 1 2 8 7 6 9 8 6 1

0 0 0 0 0 0 0 0 0 0  
1 1 1 1 1 1 1 1 1 1  
2 2 2 2 2 2 2 2 2 2  
3 3 3 3 3 3 3 3 3 3  
4 4 4 4 4 4 4 4 4 4  
5 5 5 5 5 5 5 5 5 5  
6 6 6 6 6 6 6 6 6 6  
7 7 7 7 7 7 7 7 7 7  
8 8 8 8 8 8 8 8 8 8  
9 9 9 9 9 9 9 9 9 9

Handwritten Digit Dataset MNIST: 60,000 training samples, 10,000 test samples

# Questions?

---

---

# **Neural Networks and Deep Learning**

**Dr. Jerome J. Braun**

## **This Lecture: Convolutional Neural Networks III**

Course: Neural Networks and Deep Learning  
IE 7615

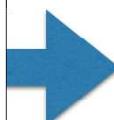
---

J. Braun

---

**Unauthorized distributing, redistributing, posting,  
and/or reposting, of the materials of this course  
(including course lectures, lecture slides, slide-sets,  
syllabi, guidelines, assignments, quizzes, exams,  
presentations, software, electronic media, etc.),  
is prohibited.**

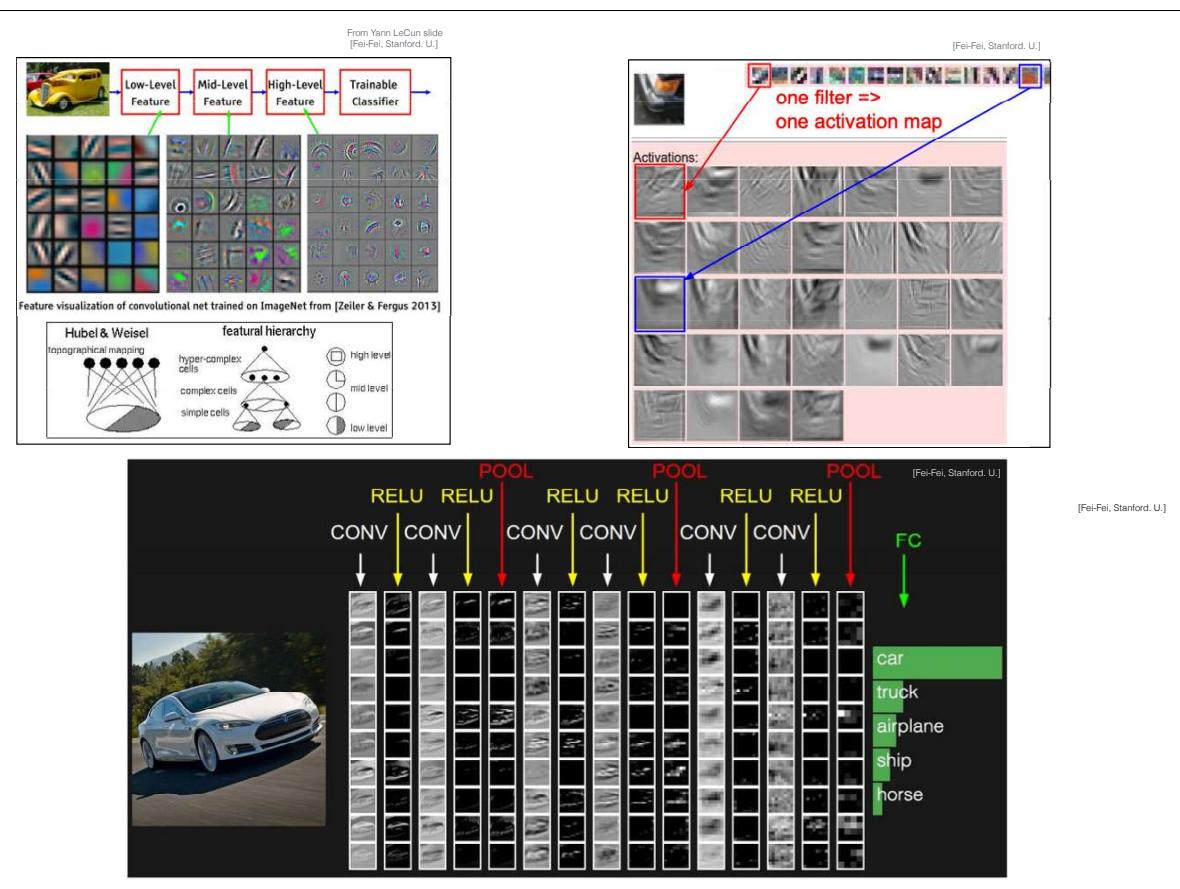
# This Lecture



- CNN architecture basics (cont.)
- CNN training

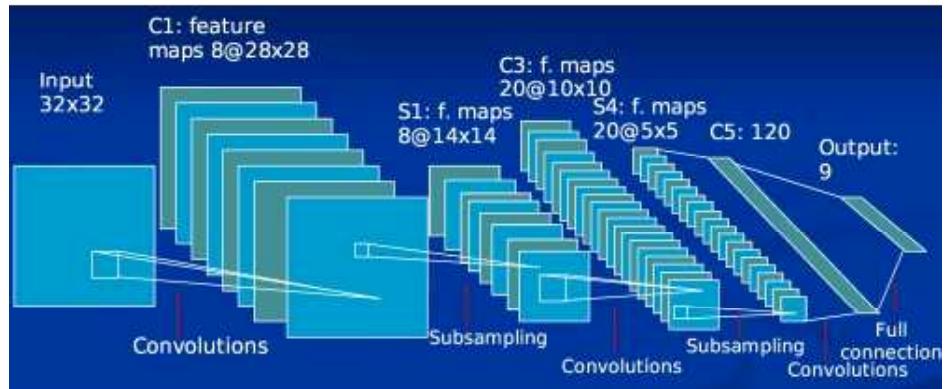
J. Braun

3



## Face Detection and Pose Estimation with Convolutional Nets

- **Training:** 52,850, 32x32 grey-level images of faces, 52,850 non-faces.
- **Each sample:** used 5 times with random variation in scale, in-plane rotation, brightness and contrast.
- **2<sup>nd</sup> phase:** half of the initial negative set was replaced by false positives of the initial version of the detector .



• New York University

Slide credit: Y. LeCun, NYU

## Face Detection and Pose Estimation with CNN

### Face Detection and Pose Estimation: Results



Yann LeCun

Credit: Y. LeCun, NYU

• New York University

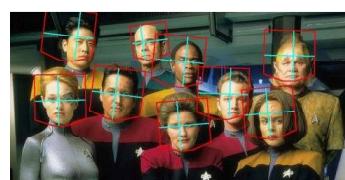
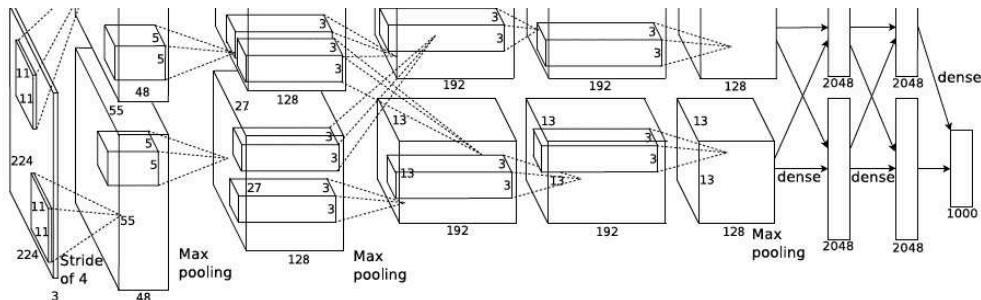


Image credit: Y. LeCun, NYU

# AlexNet

- Similar framework to LeCun'98 but:
  - Bigger model (7 hidden layers, 650,000 units, 60,000,000 params)
  - More data ( $10^6$  vs.  $10^3$  images)
  - GPU implementation (50x speedup over CPU)
    - Trained on two GPUs for a week



A. Krizhevsky, I. Sutskever, and G. Hinton,  
ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

## This Lecture

- CNN architectures (cont.)
- CNN training

# CNN Training

---

- Essentially BackProp — but with specifics particular to CNN fundamentals, e.g.,
  - Kernels (filters)
  - CNN architectural details
- CNN-oriented specifics include
  - Weight sharing
  - Separate biases (for each location of output map) or bias parameter sharing
- 

## CNN Training (cont.)

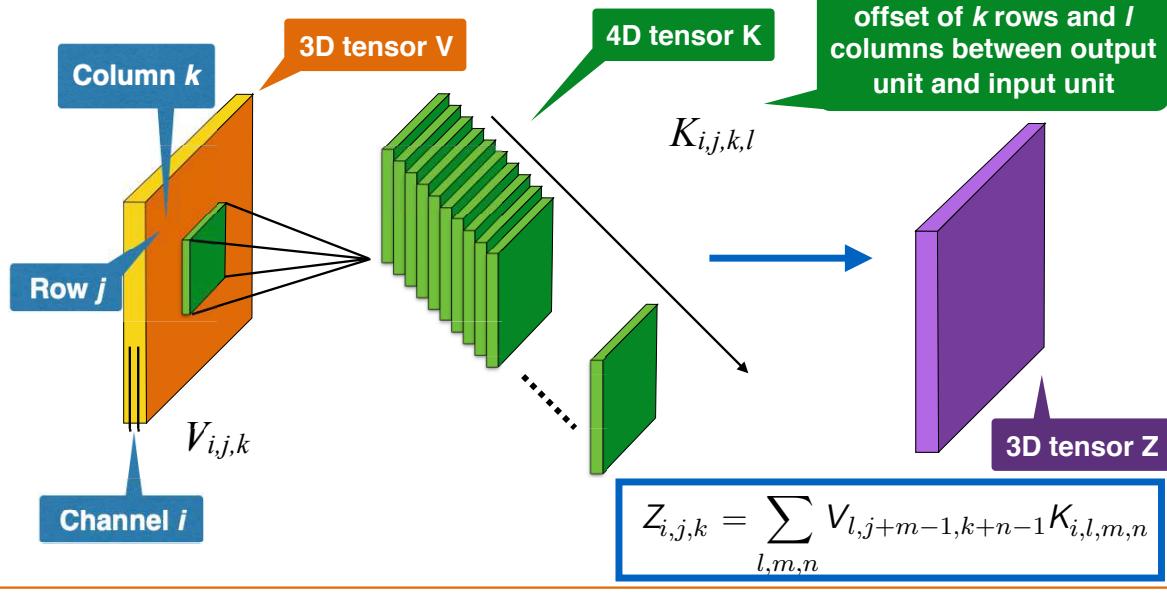
---

In general, in addition to basic convolution operations, CNN back-propagation includes

- All stages of a given CNN architecture — in particular, non-linearity operations
- Bias choice and specifics
  - For FC layers, typical to assign each unit its own bias
  - For tiled convolution, typical to share the biases with same tiling pattern as that of kernels
  - For convolutional layers, typical to:
    - Use one bias per channel of output
    - Share it across all locations within each convolution map
    - When input size is known and fixed, then possible to use separate bias at each location of output map
- Separating biases:
  - May reduce slightly statistical efficiency of model
  - But allows model to correct for differences in statistics at different locations
  - Example: When zero padding used, detector units at edge of the input receive less total input and may need larger biases
- For clarity, in formulas that follow (subsequent slides) these aspects are not included

# CNN Training (Image Context)

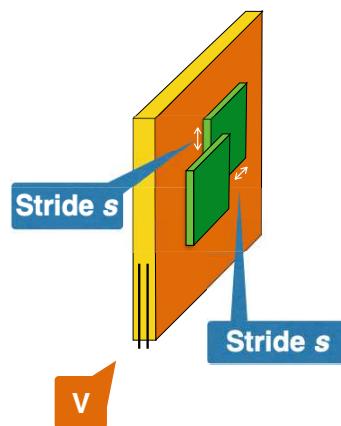
- Kernel stack in batch-mode implementations



J. Braun

11

## CNN Training (Image Context): Downsampled Convolution



$$Z_{i,j,k} = c(\mathbf{K}, \mathbf{V}, s)_{i,j,k} = \sum_{l,m,n} [V_{l,(j-1)\times s+m, (k-1)\times s+n} K_{i,l,m,n}]$$

J. Braun

12

## CNN Training (Image Context): Forward Propagation Pass

---

- Strided convolution of kernel stack  $\mathbf{K}$  applied to multi-channel image  $\mathbf{V}$ 
  - Stride  $s$

$$Z_{i,j,k} = c(\mathbf{K}, \mathbf{V}, s)_{i,j,k} = \sum_{l,m,n} [V_{l,(j-1)\times s+m, (k-1)\times s+n} K_{i,l,m,n}]$$

- Propagate through network
- Compute cost function  $J(\mathbf{V}, \mathbf{K})$ 
  - Chosen loss (cost) function  $J$

## CNN Training (Image Context): Back-Propagation Pass

---

- Receive tensor  $\mathbf{G}$ :  $G_{i,j,k} = \frac{\partial}{\partial Z_{i,j,k}} J(\mathbf{V}, \mathbf{K})$
- Compute derivatives w.r.t. weights in kernel  

$$g(\mathbf{G}, \mathbf{V}, s)_{i,j,k,l} = \frac{\partial}{\partial K_{i,j,k,l}} J(\mathbf{V}, \mathbf{K}) = \sum_{m,n} G_{i,m,n} V_{j,(m-1)\times s+k, (n-1)\times s+l}$$
- If not at bottom of network, compute gradient w.r.t.  $\mathbf{V}$ 
  - To back-propagate further down

$$\begin{aligned}
 h(\mathbf{K}, \mathbf{G}, s)_{i,j,k} &= \frac{\partial}{\partial V_{i,j,k}} J(\mathbf{V}, \mathbf{K}) \\
 &= \sum_{\substack{l,m \\ \text{s.t.} \\ (l-1)\times s+m=j}} \sum_{\substack{n,p \\ \text{s.t.} \\ (n-1)\times s+p=k}} \sum_q K_{q,i,m,p} G_{q,l,n}
 \end{aligned}$$

# Questions?

---