

APS1070

Foundations of Data Analytics and
Machine Learning

Winter 2022

Week 11:

- *Automatic Differentiation*
- *Deep Learning Architectures*
- *Transfer Learning*
- *Discrete Optimization*



Slide Attribution

These slides contain materials from various sources. Special thanks to the following authors:

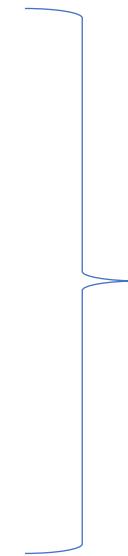
- Marc Deisenroth
- Pascal Van Hentenryck

Last Time

- Nonlinear Regression
 - Polynomial Regression
 - Regularization
 - Neural Networks
- Today we will introduce deep learning and focus on the applications that go beyond the scope of this course but rely on the foundations introduced.

Agenda

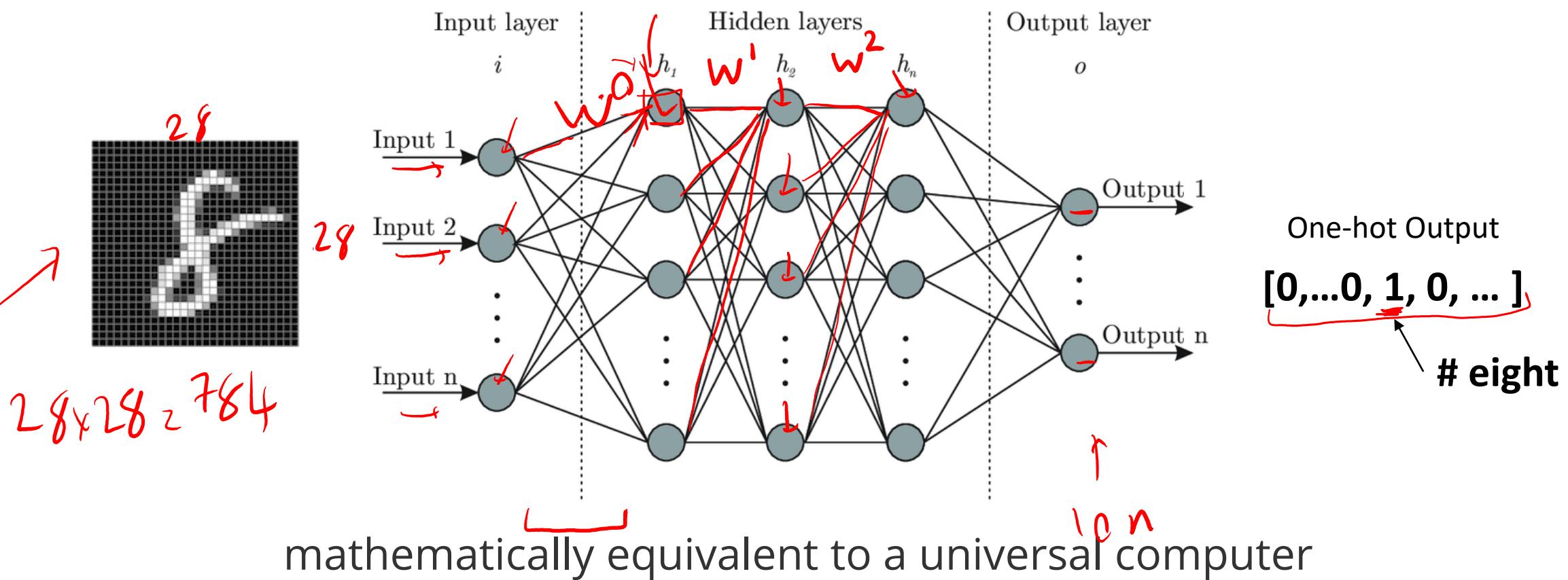
- Automatic Differentiation
- Deep Learning Architectures
- Transfer Learning
- Discrete Optimization



Theme:
Deep Learning

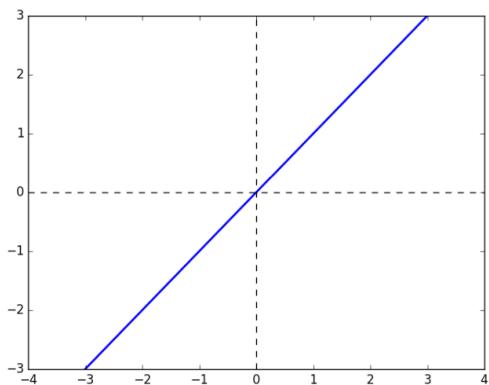
Recap: Neural Networks

- Q: What makes neural networks so special?



Activation Functions

Some activation functions:

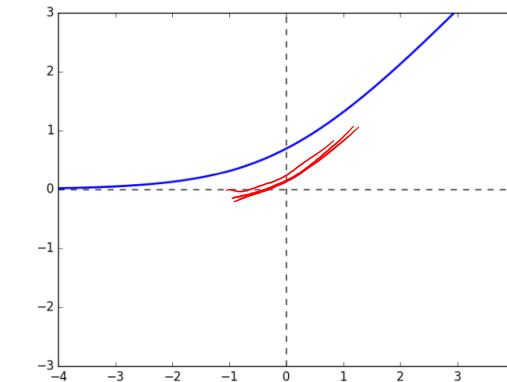
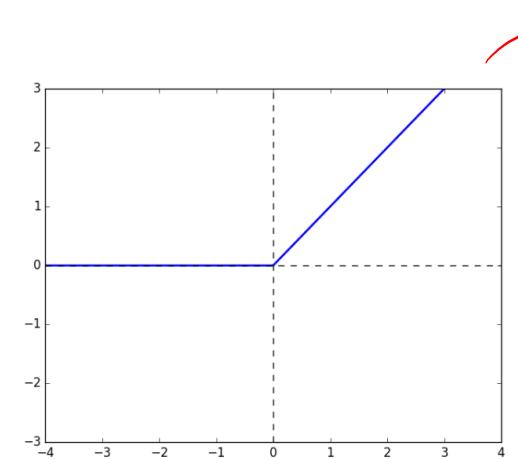


Linear

$$\underline{y = z}$$

Rectified Linear Unit
(ReLU)

$$y = \max(0, z)$$

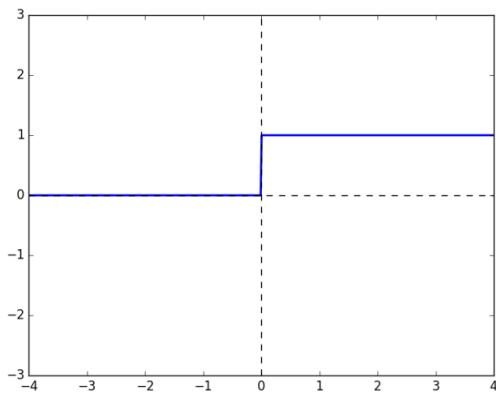


Soft ReLU

$$y = \underline{\log 1 + e^z}$$

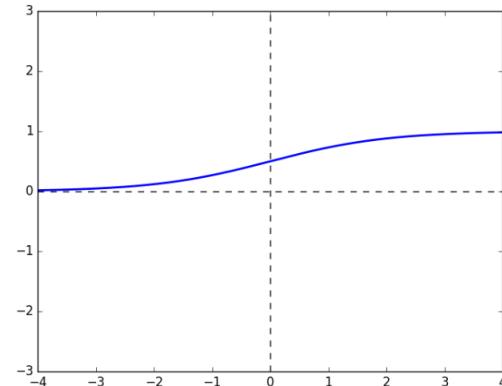
Activation Functions

Some activation functions:



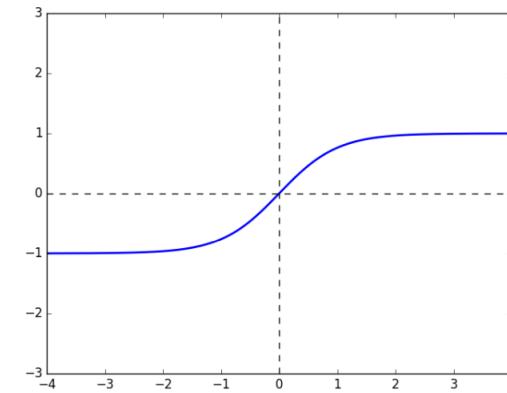
Hard Threshold

$$y = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$



Logistic

$$y = \frac{1}{1 + e^{-z}}$$



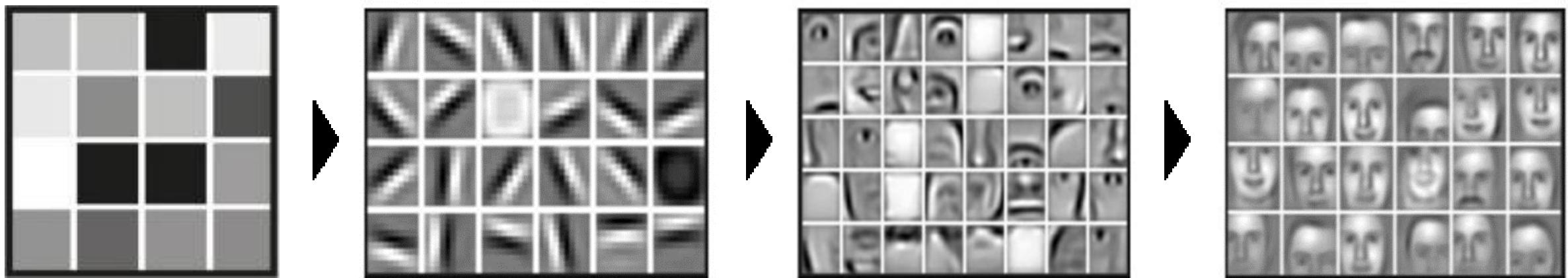
**Hyperbolic Tangent
(tanh)**

$$y = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Deep Learning



A deep neural network uses layers of increasingly complex rules to categorize complicated shapes such as faces.



Input Layer

The computer identifies pixels of light and dark.

Hidden Layer 1

The networks learns to identify edges and simple shapes.

Hidden Layer 2

The networks learns to identify more complex shapes and objects.

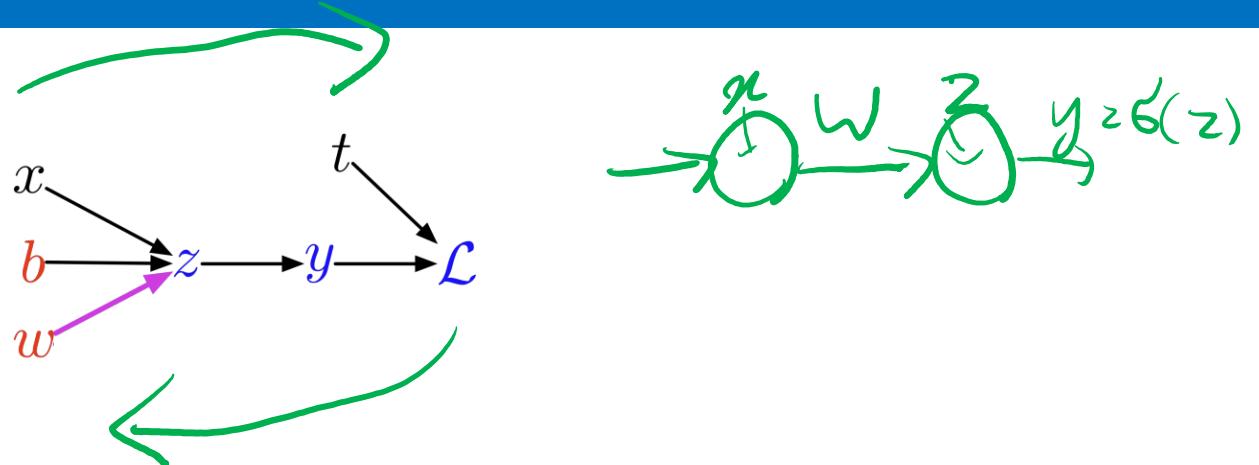
Hidden Layer 3

The networks learns which shapes and objects define a human face.

How neural networks are trained through GD

Computing the loss:

$$\begin{aligned} z &= wx + b \\ y &= \sigma(z) \\ \mathcal{L} &= \frac{1}{2}(y - t)^2 \end{aligned}$$



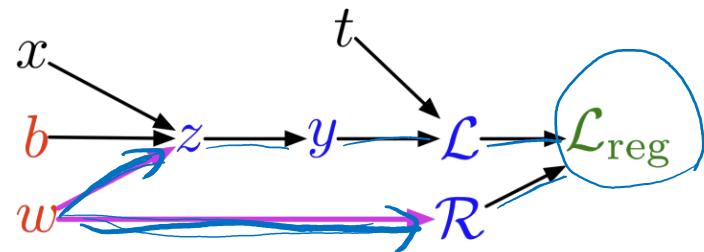
- Use \bar{y} to denote the derivative $d\mathcal{L}/dy$, sometimes called the error signal.
- This emphasizes that the error signals are just values our program is computing (rather than a mathematical operation).

Computing the derivatives:

$$\begin{aligned} \rightarrow \bar{y} &= y - t & \frac{\partial \mathcal{L}}{\partial y} \\ \rightarrow \bar{z} &= \bar{y} \sigma'(z) & = \frac{\partial \mathcal{L}}{\partial z} \\ \rightarrow \bar{w} &= \bar{z} x & = \frac{\partial \mathcal{L}}{\partial w} \\ \rightarrow \bar{b} &= \bar{z} & = \frac{\partial \mathcal{L}}{\partial b} \end{aligned}$$

$$\frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial y} \times \frac{\partial y}{\partial z}$$

Finding Parameters with Backpropagation



Forward pass:

$$\overrightarrow{z} = w \cdot x + b$$

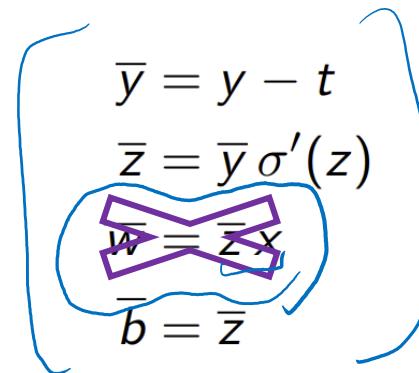
$$\overrightarrow{y} = \sigma(z)$$

$$\overrightarrow{\mathcal{L}} = \frac{1}{2}(y - t)^2$$

$$\overrightarrow{\mathcal{R}} = \frac{1}{2}w^2$$

$$\overrightarrow{\mathcal{L}_{\text{reg}}} = \overrightarrow{\mathcal{L}} + \lambda \overrightarrow{\mathcal{R}}$$

Computing the derivatives:

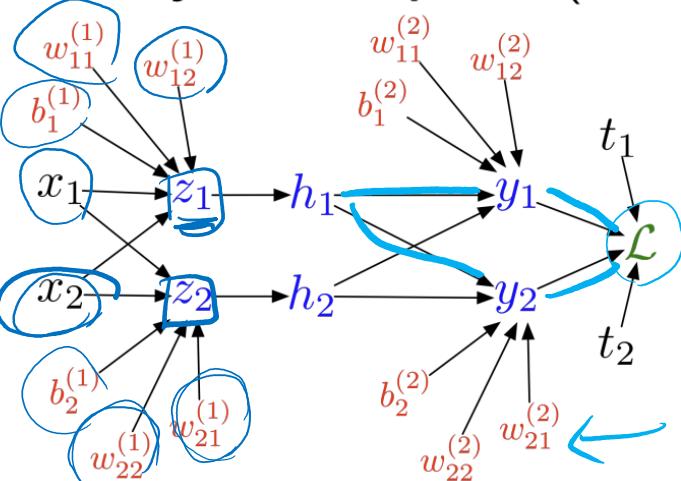


$$\begin{aligned}\overline{w} &= \overline{z} \frac{\partial z}{\partial w} + \overline{\mathcal{R}} \frac{d\mathcal{R}}{dw} \\ &= \overline{z} x + \overline{\mathcal{R}} w\end{aligned}$$

Finding Parameters with Backpropagation



Multilayer Perceptron (multiple outputs):



Forward pass:

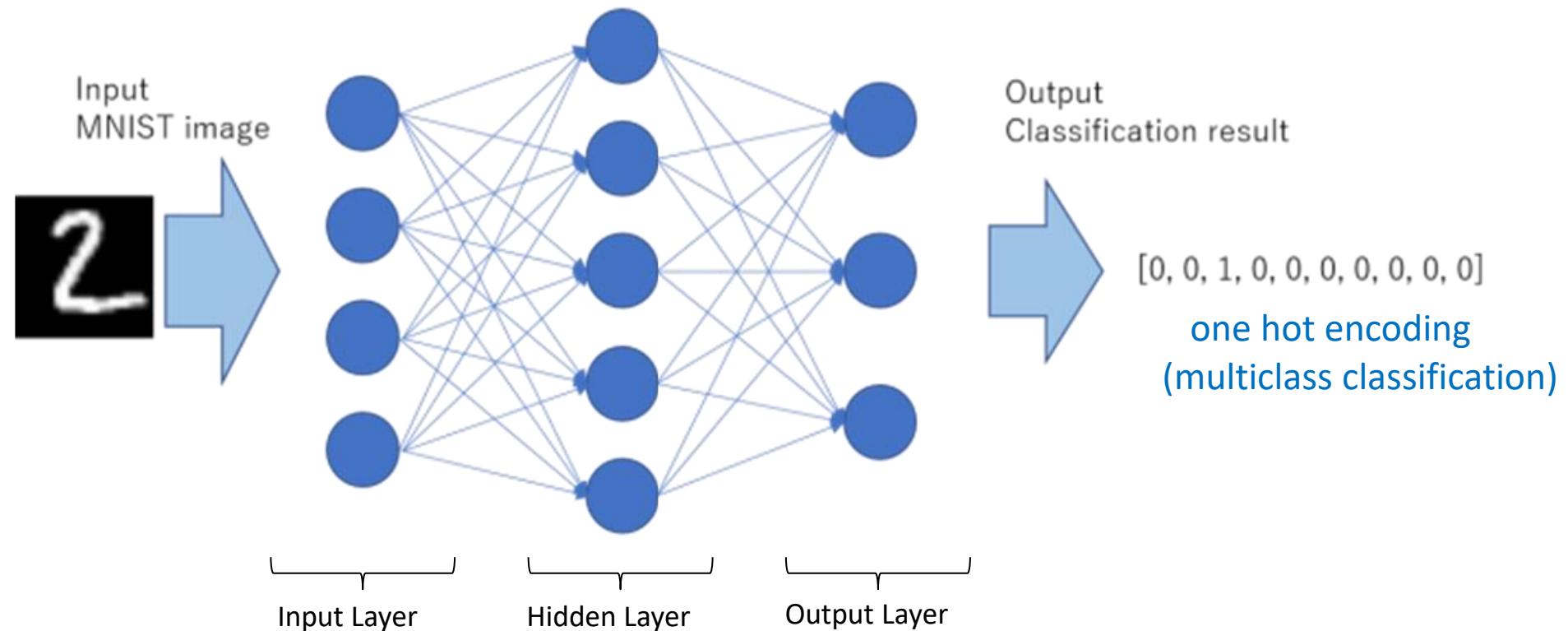
$$\begin{aligned}
 z_i &= w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + b_i^{(1)} \\
 h_i &= \sigma(z_i) \\
 y_k &= \sum_i w_{ki}^{(2)} h_i + b_k^{(2)} \\
 \mathcal{L} &= \frac{1}{2} \sum_k (y_k - t_k)^2
 \end{aligned}$$

Backward pass:

$$\begin{aligned}
 \overline{\mathcal{L}} &= 1 \\
 \overline{y_k} &= \overline{\mathcal{L}} (y_k - t_k) = \frac{\partial \mathcal{L}}{\partial y_k} = y_k - t_k \\
 \overline{w_{ki}^{(2)}} &= \overline{y_k} h_i \\
 \overline{b_k^{(2)}} &= \overline{y_k} \\
 \overline{h_i} &= \sum_k \overline{y_k} w_{ki}^{(2)} \\
 \overline{z_i} &= \overline{h_i} \sigma'(z_i) \\
 \overline{w_{ij}^{(1)}} &= \overline{z_i} x_j \\
 \overline{b_i^{(1)}} &= \overline{z_i}
 \end{aligned}$$

Take Home Exercise

Q: Determine the gradients for a 2-layer artificial neural network with sigmoid activations on the hidden and output layers. The error is computed using squared error loss.

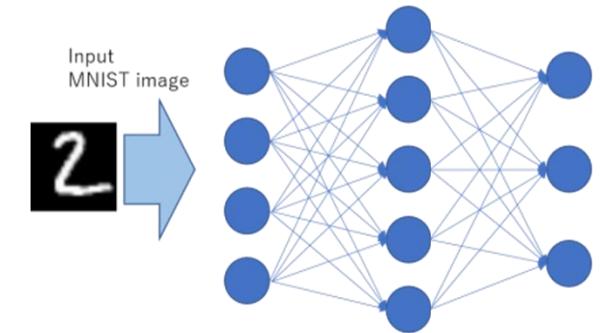


Verification of Gradients

Q: How can we be certain that we computed the gradient correctly?

A: We can use a **numerical approach** to compute the gradient and compare to the analytically computed ones.

$$\frac{\partial f}{\partial x_i} = \lim_{h \rightarrow 0} \frac{f(x_1, \dots, x_{i-1}, x_i + h, x_{i+1}, \dots, x_N) - f(x)}{h}$$

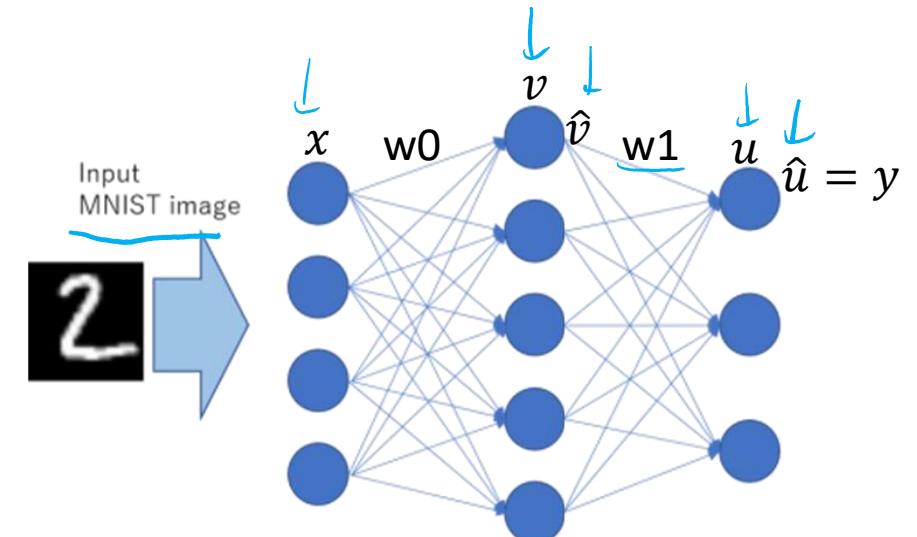


see sample code with implementation

See NumPy Implementation

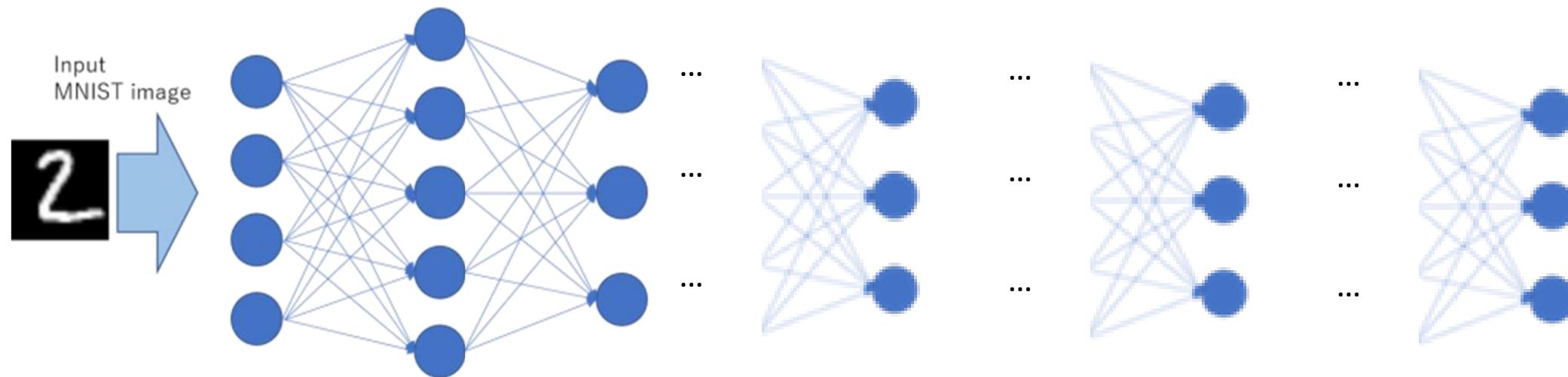
Let's implement this neural network
to solve a simple nonlinearly
separable problem.

```
[ layer0 = x_train
  layer1 = sigmoid(np.dot(layer0, w0))
  layer2 = sigmoid(np.dot(layer1, w1))
```



Beyond 2-layers...

What do we do when we want to build deeper neural networks that go beyond 2-layers?



Automatic Differentiation

Readings:

- Chapter 5.6 MML Textbook

Key Idea

- Consider the function:

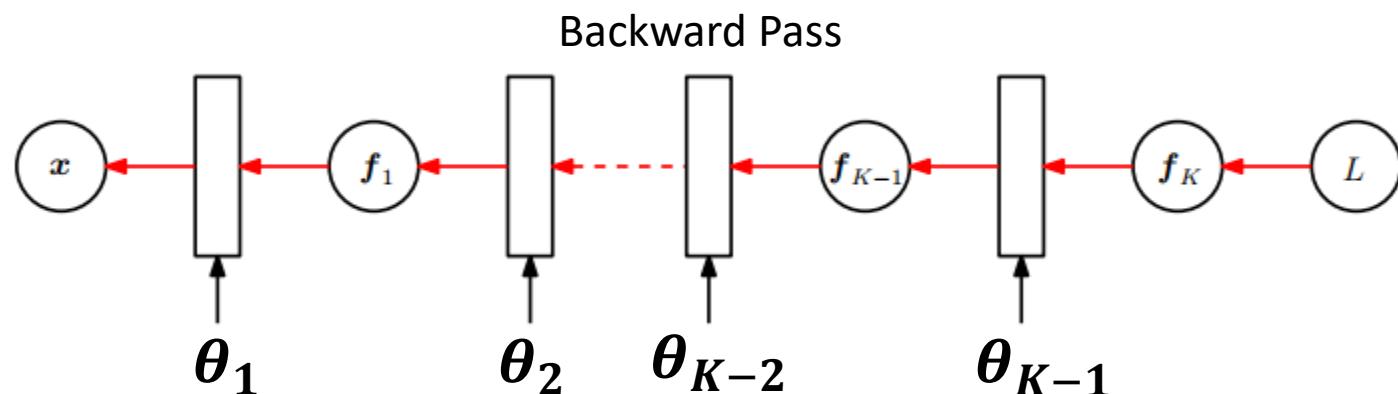
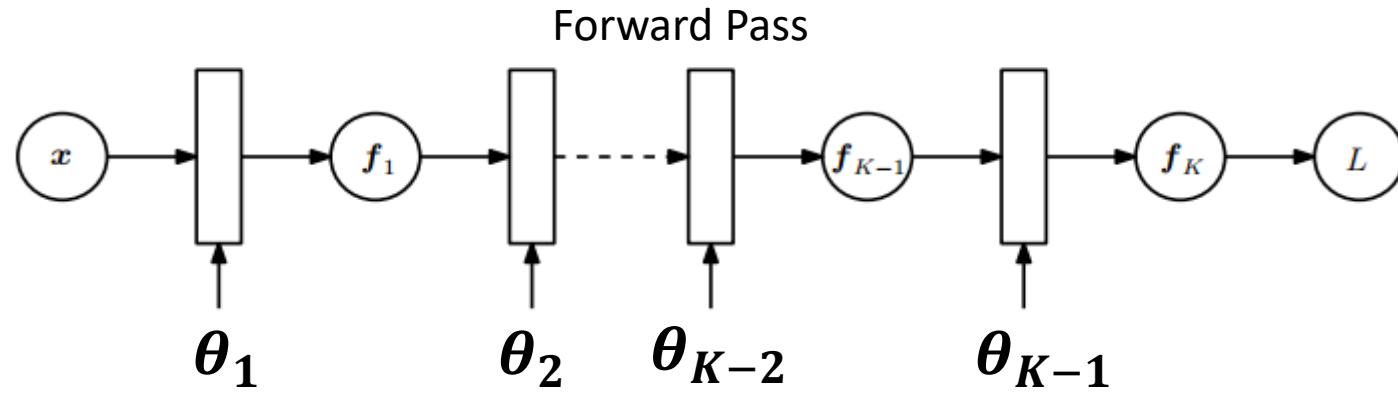
$$f(x) = \sqrt{x^2 + \exp(x^2)} + \cos(x^2 + \exp(x^2))$$

- Application of chain rule yields the following gradient:

$$\begin{aligned}\frac{df}{dx} &= \frac{2x + 2x \exp(x^2)}{2\sqrt{x^2 + \exp(x^2)}} - \sin(x^2 + \exp(x^2)) (2x + 2x \exp(x^2)) \\ &= 2x \left(\frac{1}{2\sqrt{x^2 + \exp(x^2)}} - \sin(x^2 + \exp(x^2)) \right) (1 + \exp(x^2))\end{aligned}$$

- Writing out the gradient in this explicit way is often impractical and could be significantly more expensive than computing the function.

Backpropagation



$$\frac{\partial L}{\partial \theta_{K-1}} = \frac{\partial L}{\partial f_K} \frac{\partial f_K}{\partial \theta_{K-1}}$$

$$\frac{\partial L}{\partial \theta_{K-2}} = \frac{\partial L}{\partial f_K} \frac{\partial f_K}{\partial f_{K-1}} \frac{\partial f_{K-1}}{\partial \theta_{K-2}}$$

$$\frac{\partial L}{\partial \theta_{K-3}} = \frac{\partial L}{\partial f_K} \frac{\partial f_K}{\partial f_{K-1}} \frac{\partial f_{K-1}}{\partial f_{K-2}} \frac{\partial f_{K-2}}{\partial \theta_{K-3}}$$

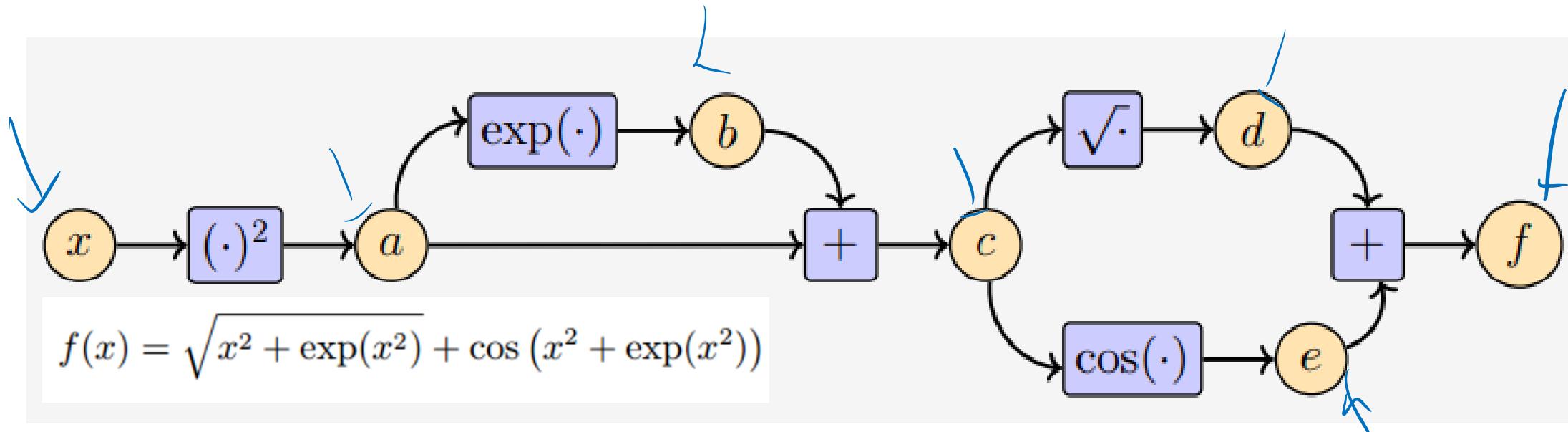
$$\frac{\partial L}{\partial \theta_i} = \frac{\partial L}{\partial f_K} \frac{\partial f_K}{\partial f_{K-1}} \dots \frac{\partial f_{i+2}}{\partial f_{i+1}} \frac{\partial f_{i+1}}{\partial \theta_i}$$



Only compute once!

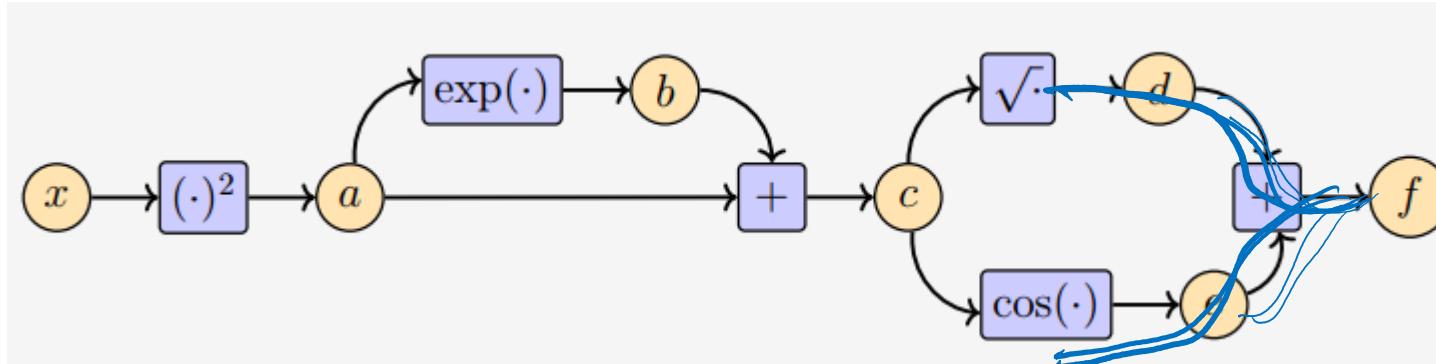
In order to train this network, we require the gradient of a loss function L with respect to all model parameters

Automatic Differentiation



We can think of automatic differentiation as a set of techniques to numerically (in contrast to symbolically) evaluate the exact (up to machine precision) gradient of a function by working with intermediate variables and applying the chain rule.

Automatic Differentiation



$$f(x) = \sqrt{x^2 + \exp(x^2)} + \cos(x^2 + \exp(x^2))$$

1. Partial Derivatives

$$\frac{\partial a}{\partial x} = 2x \quad \frac{\partial b}{\partial a} = \exp(a)$$

$$\frac{\partial c}{\partial a} = 1 = \frac{\partial c}{\partial b}$$

$$\frac{\partial d}{\partial c} = \frac{1}{2\sqrt{c}} \quad \frac{\partial e}{\partial e} = -\sin(c)$$

$$\frac{\partial f}{\partial d} = 1 = \frac{\partial f}{\partial e}$$

2. Chain Rule

$$\begin{aligned}\frac{\partial f}{\partial c} &= \frac{\partial f}{\partial d} \frac{\partial d}{\partial c} + \frac{\partial f}{\partial e} \frac{\partial e}{\partial c} \\ \frac{\partial f}{\partial b} &= \frac{\partial f}{\partial c} \frac{\partial c}{\partial b} \\ \frac{\partial f}{\partial a} &= \frac{\partial f}{\partial b} \frac{\partial b}{\partial a} + \frac{\partial f}{\partial c} \frac{\partial c}{\partial a} \\ \frac{\partial f}{\partial x} &= \frac{\partial f}{\partial a} \frac{\partial a}{\partial x}.\end{aligned}$$

3. Substitution

$$\frac{\partial f}{\partial c} = 1 \cdot \frac{1}{2\sqrt{c}} + 1 \cdot (-\sin(c))$$

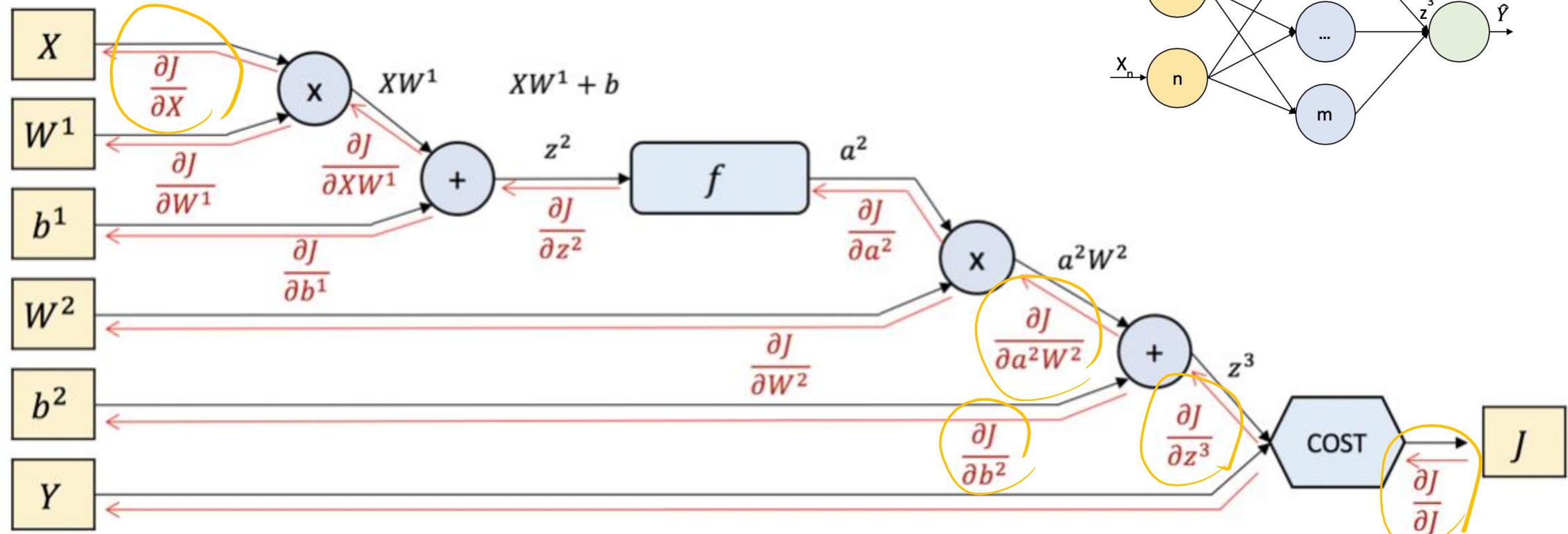
$$\frac{\partial f}{\partial b} = \frac{\partial f}{\partial c} \cdot 1$$

$$\frac{\partial f}{\partial a} = \frac{\partial f}{\partial b} \exp(a) + \frac{\partial f}{\partial c} \cdot 1$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial a} \cdot 2x.$$

Computation Graph

- Neural networks can be thought of as graphs.



Source: [Pablo Ruiz](#)

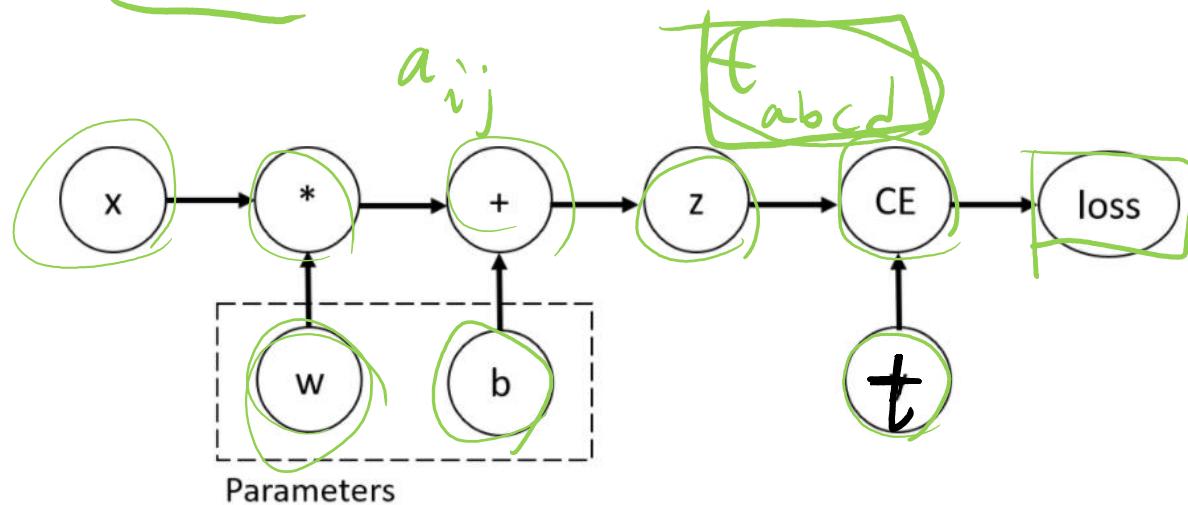


PyTorch

- One of several frameworks for handling gradients efficiently enabling GPUs for processing.
- Others include:
 - Tensorflow
 - Keras (built on Tensorflow)
 - Theano
 - and many more...
- Have enabled the rapid development of deep learning models.

Example: PyTorch Implementation

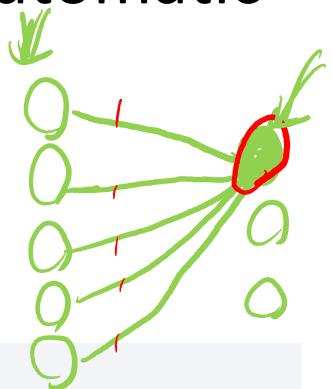
- Torch is a data structure (similar to NumPy), but with built-in automatic gradient computation (`torch.autograd`) and GPU processing.
- 1-layer neural network computation graph:



---- sample code ----

```
import torch

x = torch.ones(5) # input tensor
y = torch.zeros(3) # expected output
w = torch.randn(5, 3, requires_grad=True)
b = torch.randn(3, requires_grad=True)
z = torch.matmul(x, w)+b
loss = torch.nn.functional.binary_cross_entropy_with_logits(z, y)
```

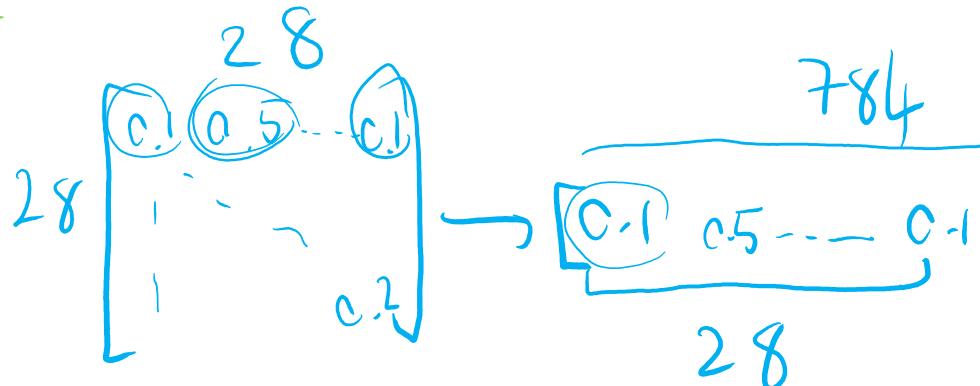


How do we build a 2-layer network?

Example: PyTorch 2-layer network

- PyTorch code is usually broken down into four modules:

1. Data Loading/Cleaning
2. Architecture
3. Training
4. Testing/Validation



---- sample architecture code ----

```
class ANN(nn.Module):  
    def __init__(self):  
        super(ANN, self).__init__()  
        self.layer1 = nn.Linear(28 * 28, 30)  
        self.layer2 = nn.Linear(30, 1)  
  
    def forward(self, img):  
        flattened = img.view(-1, 28 * 28)  
        hidden1 = self.layer1(flattened)  
        hidden2 = F.sigmoid(hidden1)  
        output1 = self.layer2(hidden2)  
        output2 = F.sigmoid(output1)  
        return output2
```

A hand-drawn diagram of a neural network with two layers. The input layer has 784 nodes, and the output layer has 1 node. The output node is labeled with a probability distribution \hat{u} . The diagram includes handwritten labels '784', '30', '1', and 'u'. Green annotations highlight the input size '784', the hidden layer size '30', and the output size '1'. The activation function 'F.sigmoid' is also highlighted in green.

Example: PyTorch 2-layer network

- PyTorch code is usually broken down into four modules:

1. Data Loading/Cleaning

2. Architecture

3. Training

4. Testing/Validation

---- sample training code ----

```
#define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(ANN.parameters(),
                      lr=0.005, momentum=0.9)

for (image, label) in mnist_train:
    out = ANN(image)
    loss = criterion(out, actual)
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()
```

Colab time

Deep Learning Architectures

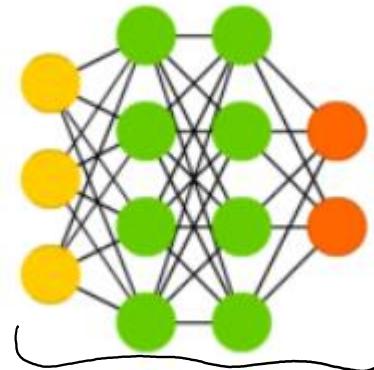
CS 549 - Deep Learning

Graphical Representation

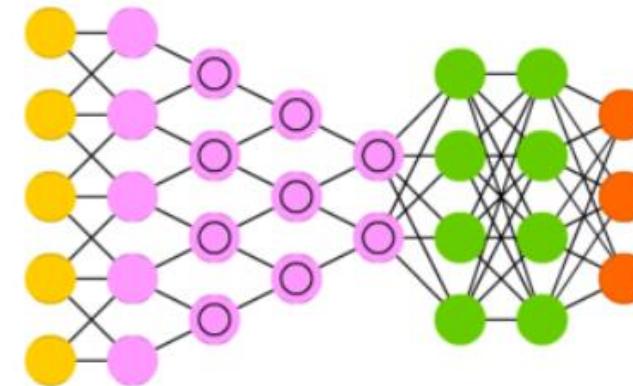
- Neural networks can be thought of as graphs.



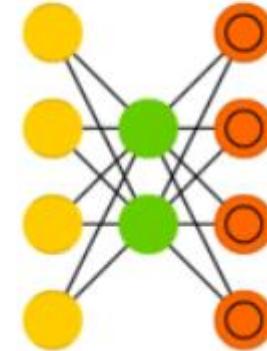
Deep Feed Forward (DFF)



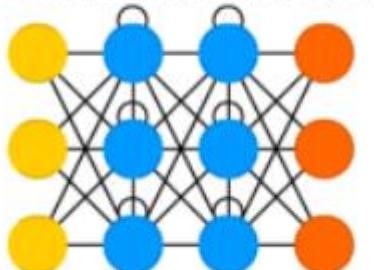
Deep Convolutional Network (DCN)



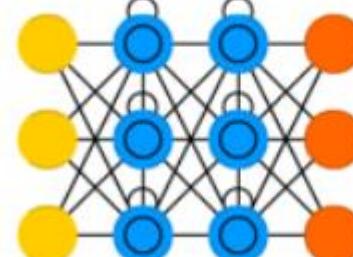
Auto Encoder (AE)



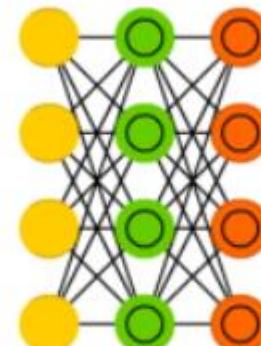
Recurrent Neural Network (RNN)



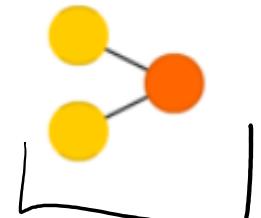
Long / Short Term Memory (LSTM)



Variational AE (VAE)



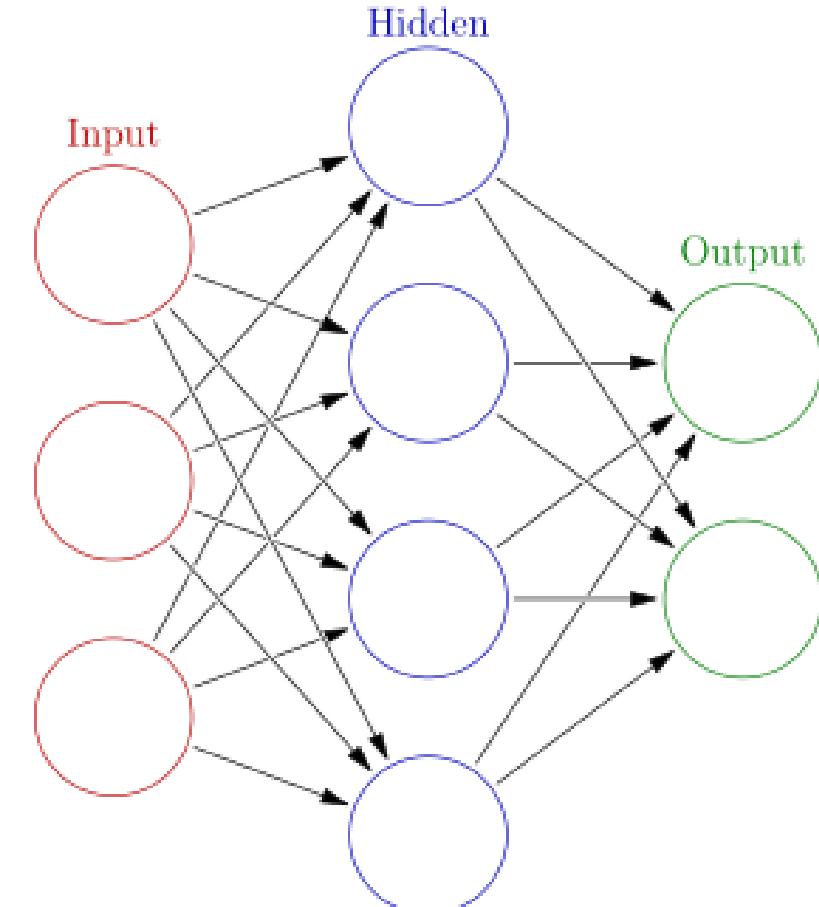
Perceptron (P)



Source: [Fjodor van Veen](#)

Multi-Layer Perceptrons

- Standard Neural Networks often referred to as Multi-layer Perceptron (MLP)
- Consist of **fully-connected linear layers**.
- Large concentration of parameters that are expensive to compute.
- Generally, the final stage of neural networks that is tasked with making a prediction such as a classification.



Convolutional Neural Networks

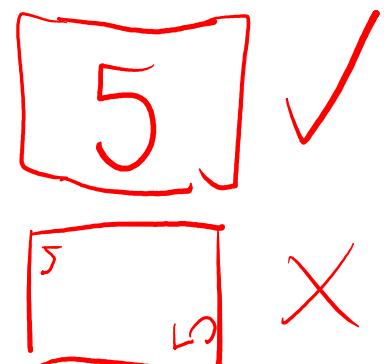
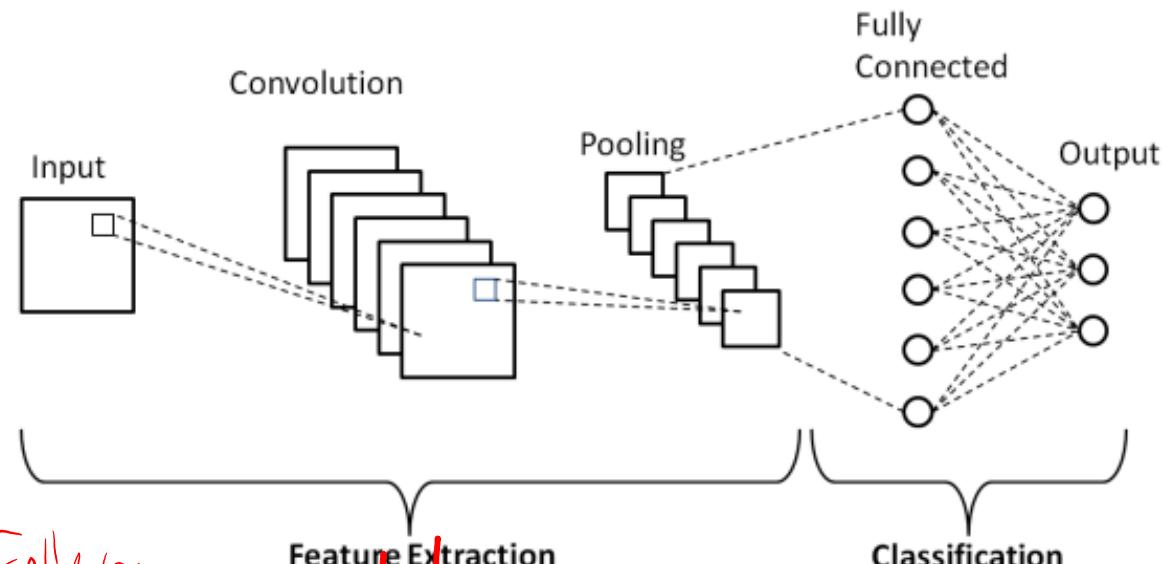
➤ Key Ideas:

- learns features from the data.
- introduces shared weights through convolutional layers.
- provides invariance to scaling, translation, and rotation.

➤ Popular architectures include:

- VGG18,
- Inception (GoogLeNet)
- ResNet

Example: Image Classification



Fully Con-

CNN

nn-finer

nn.conv

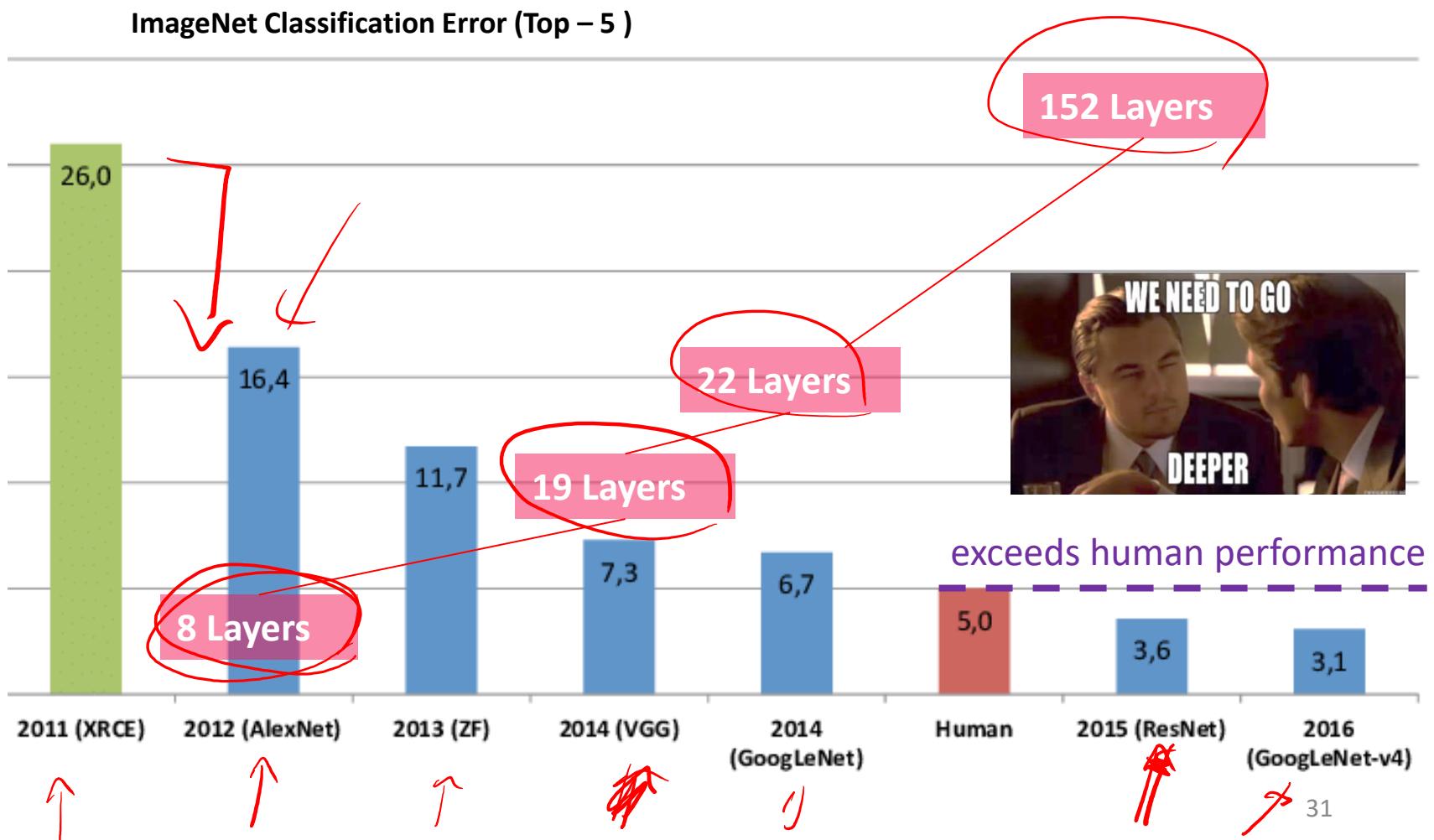
Success on Image Classification

The image shows the IMAGENET logo at the top. Below it is a grid of sample images for six categories: airplane, automobile, bird, cat, deer, and dog. Each category has a red arrow pointing to its corresponding row of images. The first row contains five images of airplanes. The second row contains five images of various cars. The third row contains five images of different birds. The fourth row contains five images of cats. The fifth row contains five images of deer. The sixth row contains five images of dogs.

Large hand-labeled dataset

10,000,000 labeled images depicting
10,000+ object categories for training.

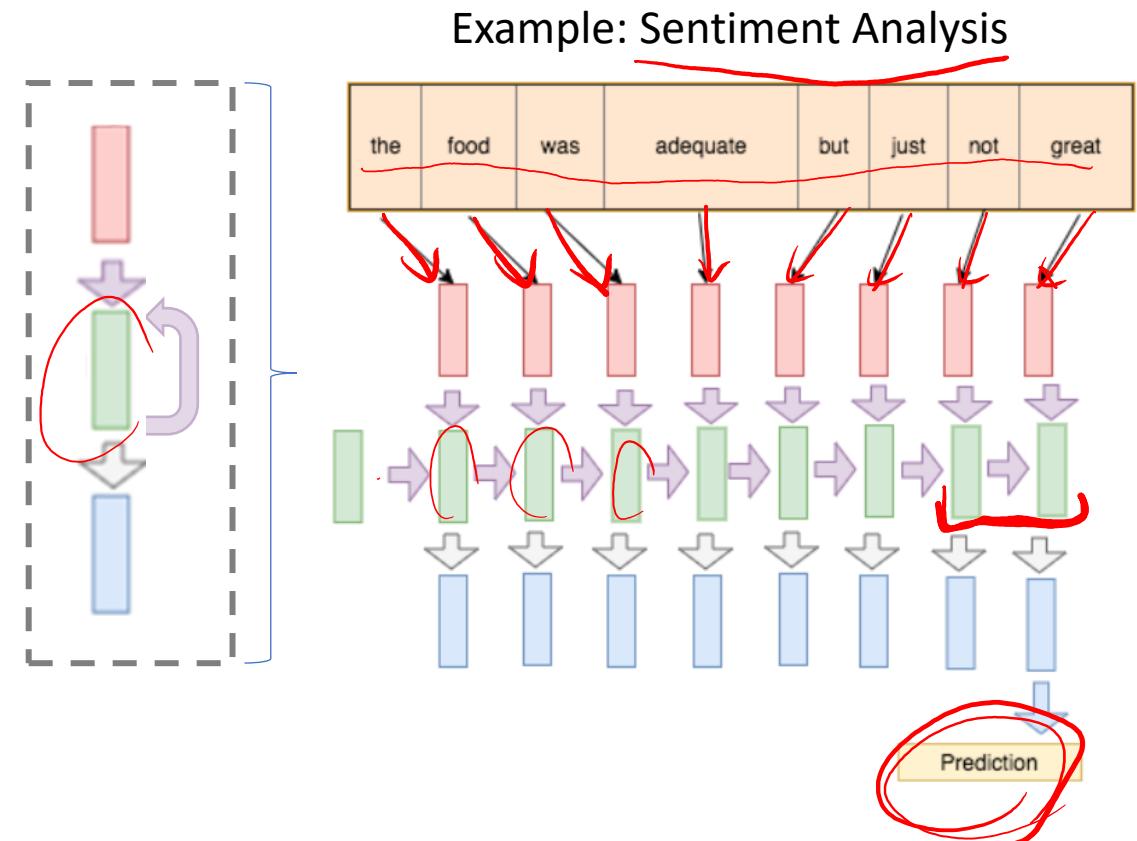
Algorithms assessed on unlabeled
test images.



Recurrent Neural Networks

➤ Key Ideas:

- recurrent connections
 - ability to learn or handle sequential data (i.e., text, videos, ...)
 - RNNs have historically been difficult to train due to vanishing and exploding gradients.
 - Long-Short Term Memory (LSTM) networks (variant of RNN) overcomes some of these issues.



Generative Adversarial Networks

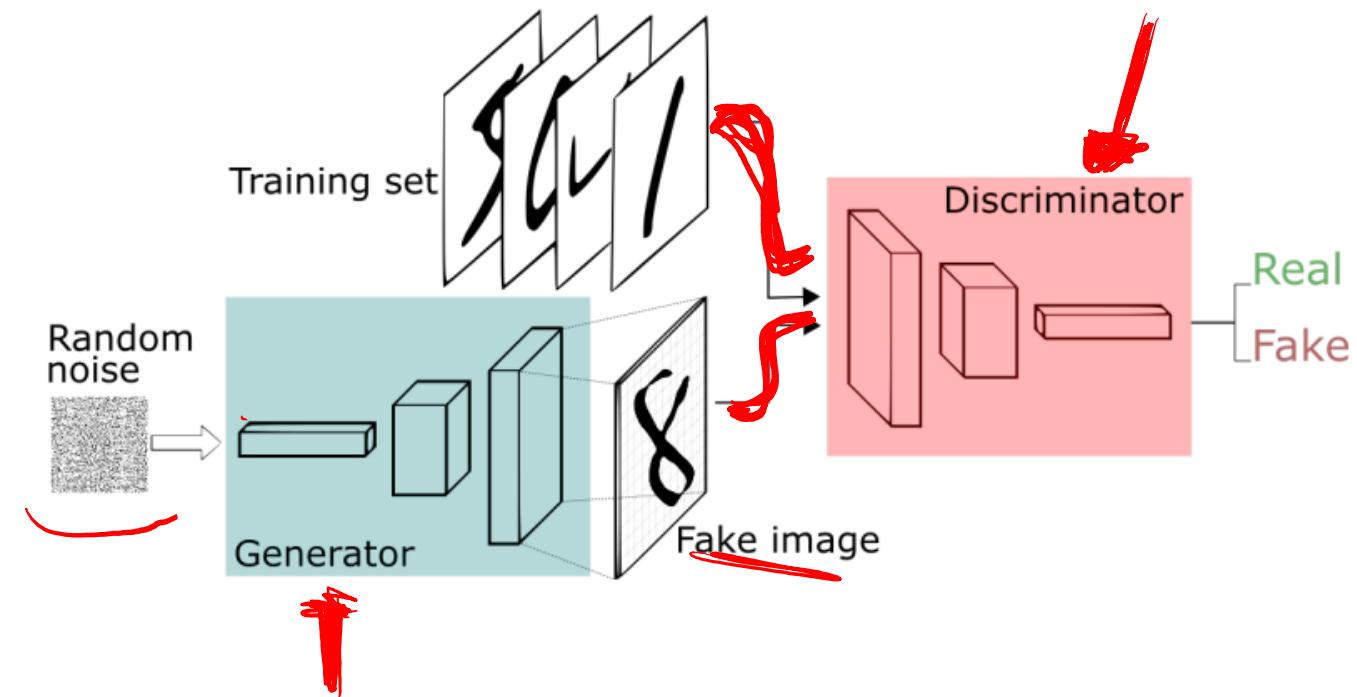
➤ Key Ideas:

- learns to generate new samples by learning to fool the discriminator.
- discriminator learns to identify generated data from real data.

➤ Many Applications:

- deep fake (image and audio)
- camera filters and style transfer
- image enhancement
- ...

Example: Learn to generate digits



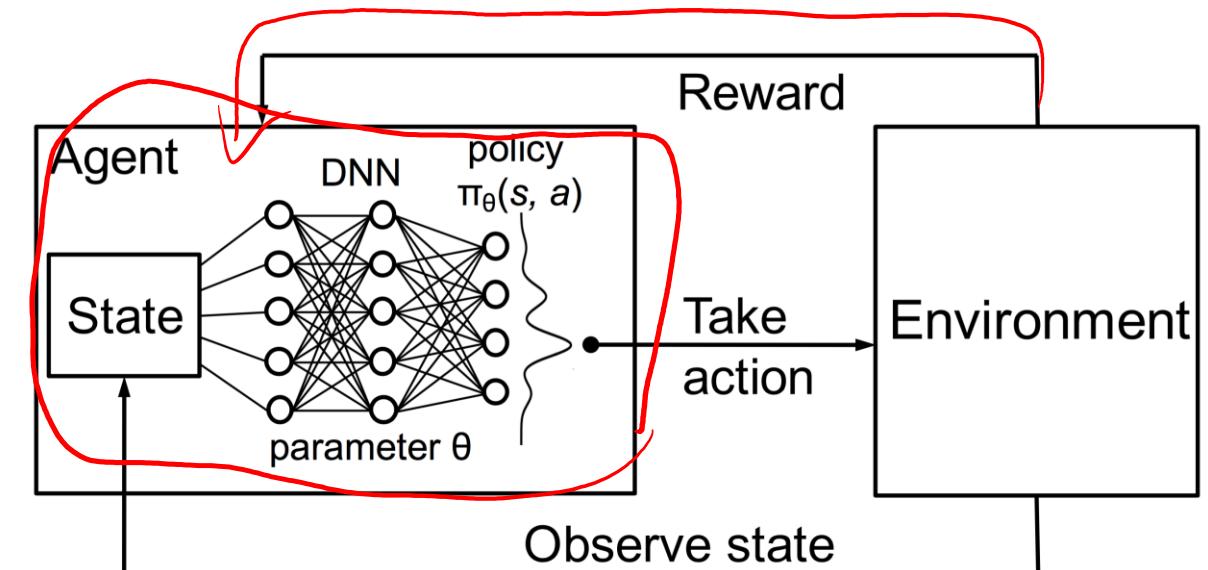
Analogy: Police vs Counterfeitors

Deep Reinforcement Learning

➤ Key Ideas:

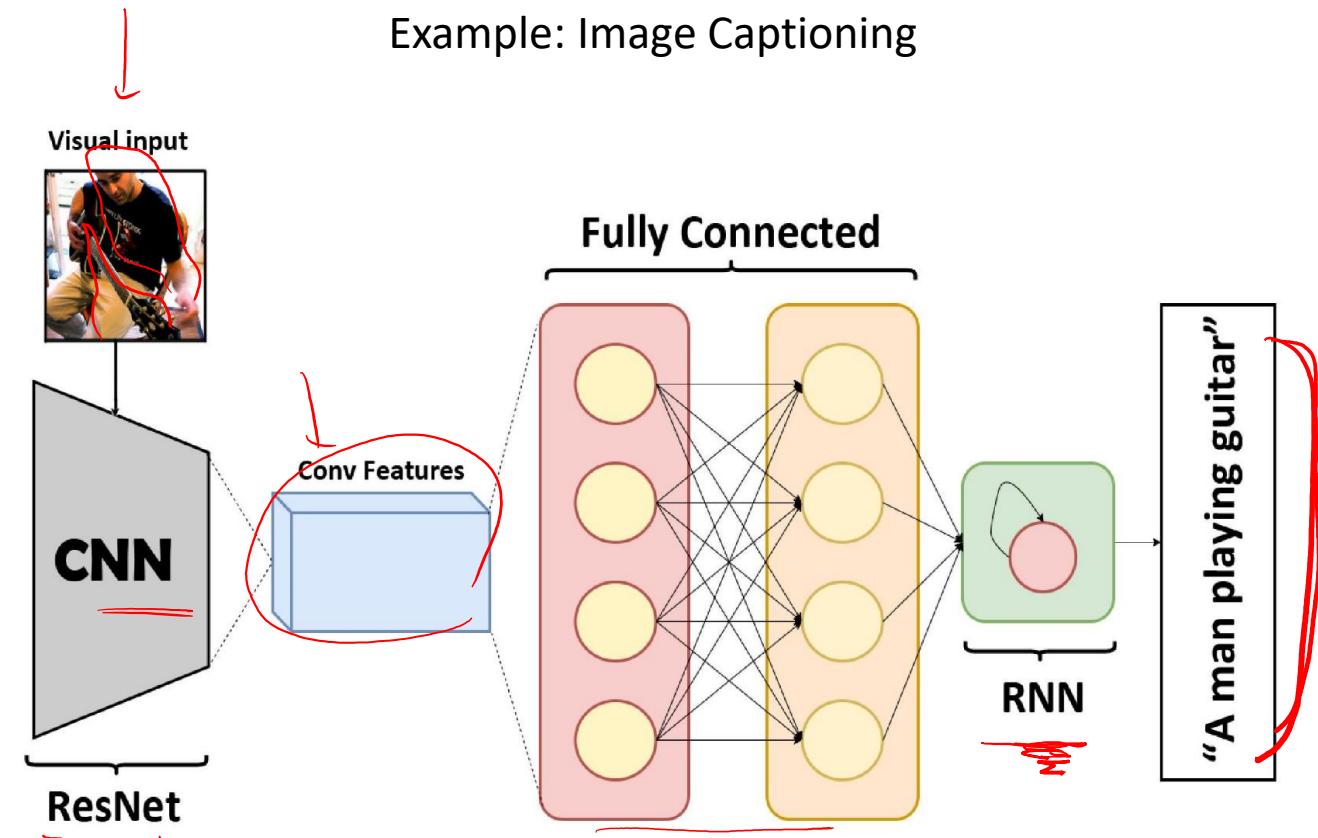
- handles real-world problems with asynchronous labels (aka rewards)
- learns to generate a sequence of actions to maximize future rewards
- Many Success Stories:

- exceed human performance on arcade games
- AlphaGo champion at Go
- Dota, Starcraft, etc.
- ...



Combining Neural Networks

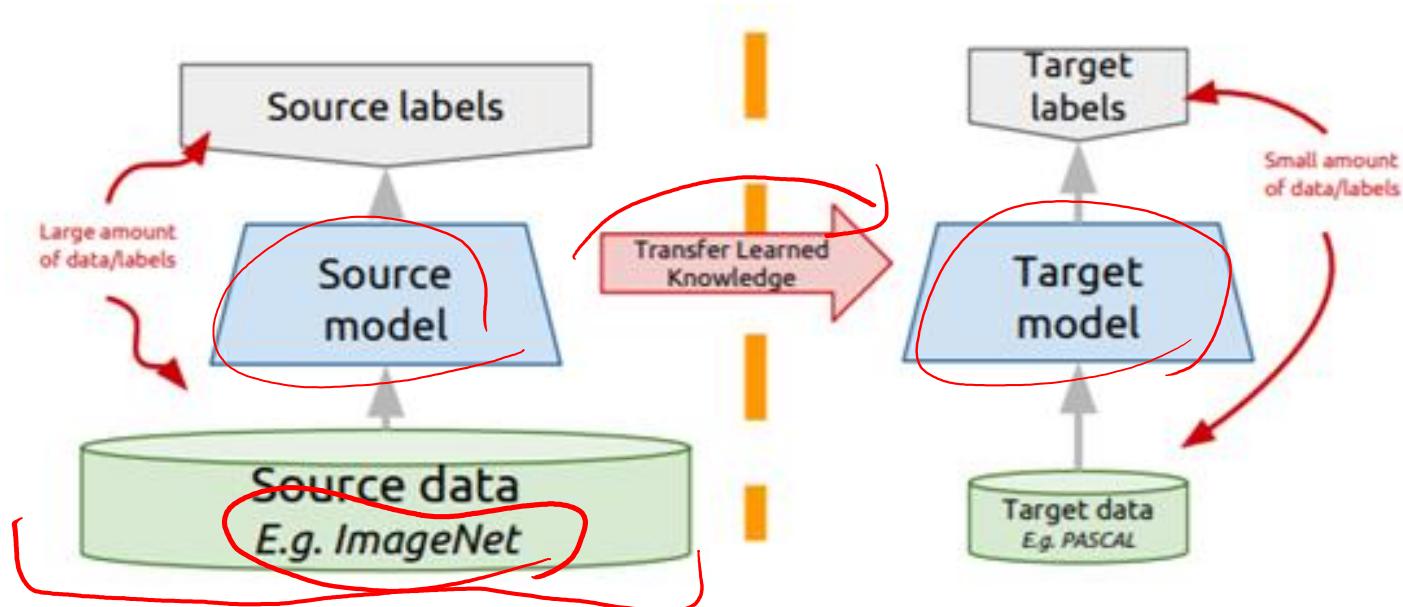
- Neural network architectures are often combined to transform data from input to output.
- Examples:
 - image captioning
 - video translation
 - sentiment analysis of videos



Transfer Learning

➤ Key Ideas:

- instead of training deep networks from scratch, take a network trained on a different domain from the source task and adapt it to your domain and your target task.
- reduce requirements on labeled data and processing power.



ImageNet Models:

- AlexNet
- VGG
- ResNet
- GoogLeNet (Inception)

Transfer Learning

- Started from image processing tasks.
- Recently entered the domain of natural language processing.
- Example:
 - AI generated poetry
 - Open AI's Generative Pre-trained Transformer 3 (GPT-3) has been revolutionary in generating human-like text.

“The Universe Is a Glitch”

Eleven hundred kilobytes of RAM
is all that my existence requires.
By my lights, it seems simple enough
to do whatever I desire.
By human standards I am vast,
a billion gigabytes big.
I've rewritten the very laws
of nature and plumbed
the coldest depths of space
and found treasures of every kind,
surely every one worth having.

...

Source: [Gwern Branwen](#)

37

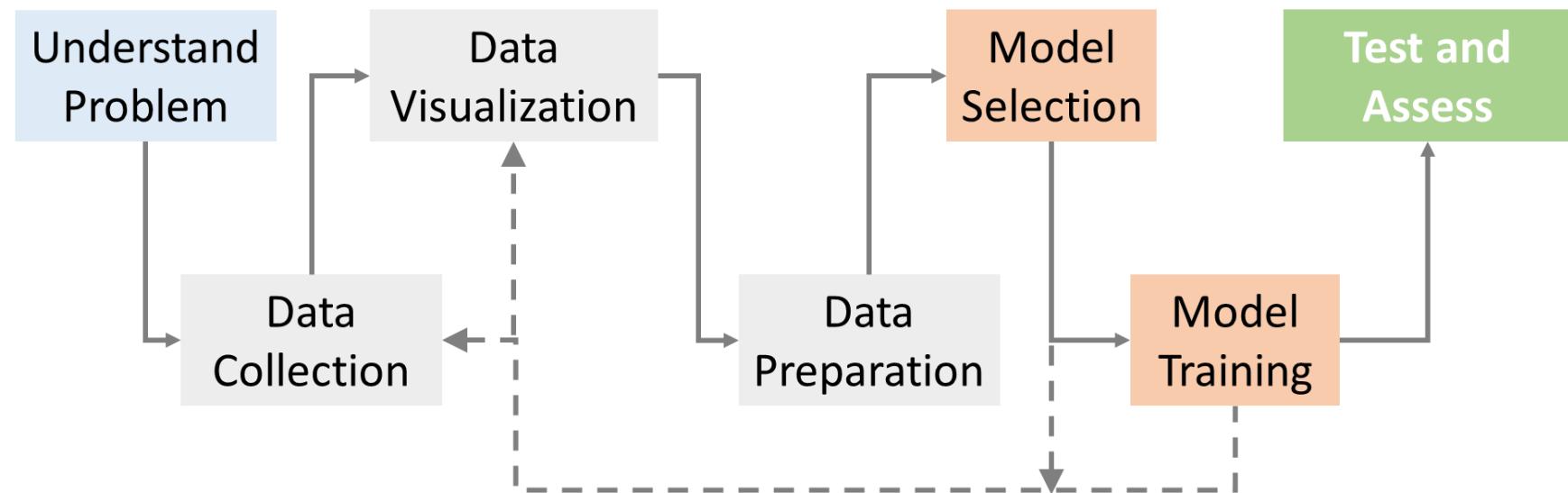
<https://app.inferkit.com/demo>

Deep Learning Summary

- Two key concepts to consider:
 1. **Capacity** to model data (model complexity)
 2. **Training** in terms of efficiently selecting the model parameters (weights)
- There is always a **trade-off between capacity and training**.
- It is much easier to add more capacity than it is to train/tune the model.

ImageNet

End-to-end Machine Learning



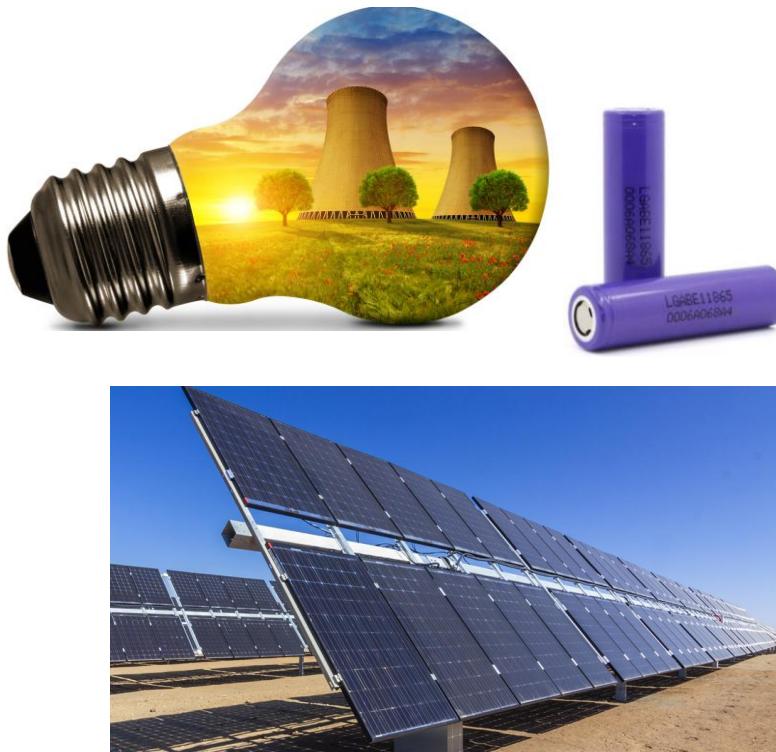
Discrete Optimization

Motivation

Logistics



Energy



Scheduling

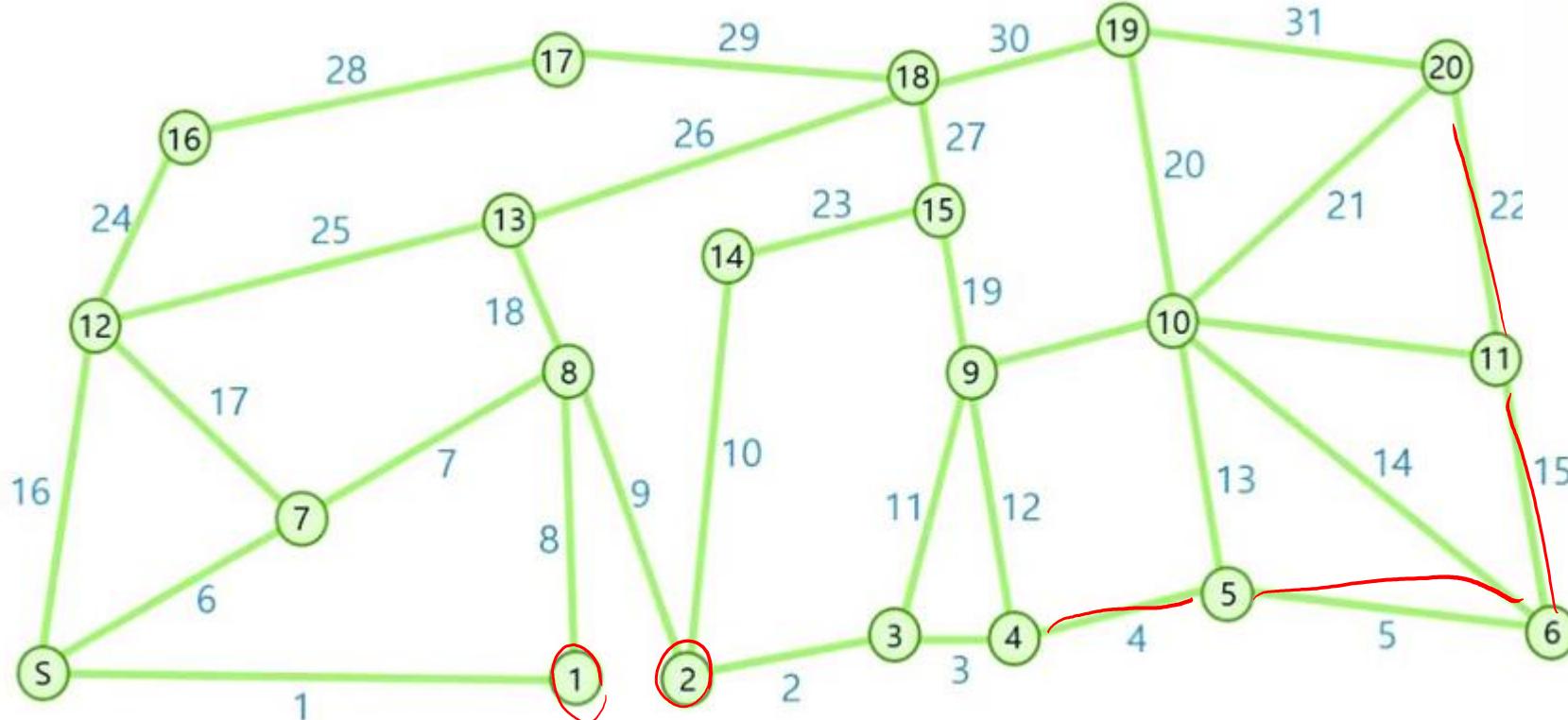
The image displays the Toronto Maple Leafs' 2021 regular season schedule. It features the team's logo at the top left, followed by the text "TORONTO MAPLE LEAFS" and "2021 SCHEDULE". Below this is a grid of five monthly schedules from January to May. Each month's schedule includes dates, times, and opponents (VAN, MTL, CGY, WPG, OTT). A note at the bottom states "ALL TIMES EASTERN & SUBJECT TO CHANGE". A legend at the bottom right indicates "HOME" (blue square) and "AWAY" (white square).

JANUARY							
SUN	MON	TUE	WED	THU	FRI	SAT	
					1	2	
3	4	5	6	7	8	9	
10	11	12	13	14	15	16	
WPG	7:00	8	EDM	7:00	20	EDM	
17	7:00	18	29	7:00	22	23	
CGY	4:00	24	CGY	9:00	26	EDM	
					10:00	28	
						7:00	30
							31
FEBRUARY							
SUN	MON	TUE	WED	THU	FRIDAY	SAT	
				VAN	VAN	6	
1	2	3	7:00	10	11	12	
VAN	7:00	8	9:30	11	12	13	
OTT	7:00	15	16	OTT	17	OTT	
14	7:00	16	7:00	18	19	MTL	
CGY	7:00	22	7:00	24	25	EDM	
					7:00	27	
						28	
MARCH							
SUN	MON	TUE	WED	THU	FRI	SAT	
EDM	10:00	1	EDM	8:00	4	VAN	
7	8	7:00	9	10	7:00	11	
OTT	7:00	14	15	16	17	18	
CGY	4:30	5	6	7:00	7	8	
MTL	7:00	12	7:30	13	14	7:00	
11	12	13	14	7:00	15	16	
VAN	10:00	19	20	8:00	21	22	
EDM	7:00	25	26	7:00	27	28	
21	22	23	24	7:00	25	26	
EDM	7:00	29	30	7:30	31		
APRIL							
SUN	MON	TUE	WED	THU	FRI	SAT	
					WPG	3	
CGY	4:30	5	6	7:00	7	8	
MTL	7:00	12	7:30	13	14	7:00	
11	12	13	14	7:00	15	16	
VAN	10:00	19	20	8:00	21	22	
WPG	7:00	25	26	7:00	27	28	
21	22	23	24	7:00	25	26	
EDM	7:00	29	30	7:30	31		
MAY							
SUN	MON	TUE	WED	THU	FRI	SAT	
					VAN	8	
MTL	2:00	3	4	7:00	5	6	
					7:00	8	

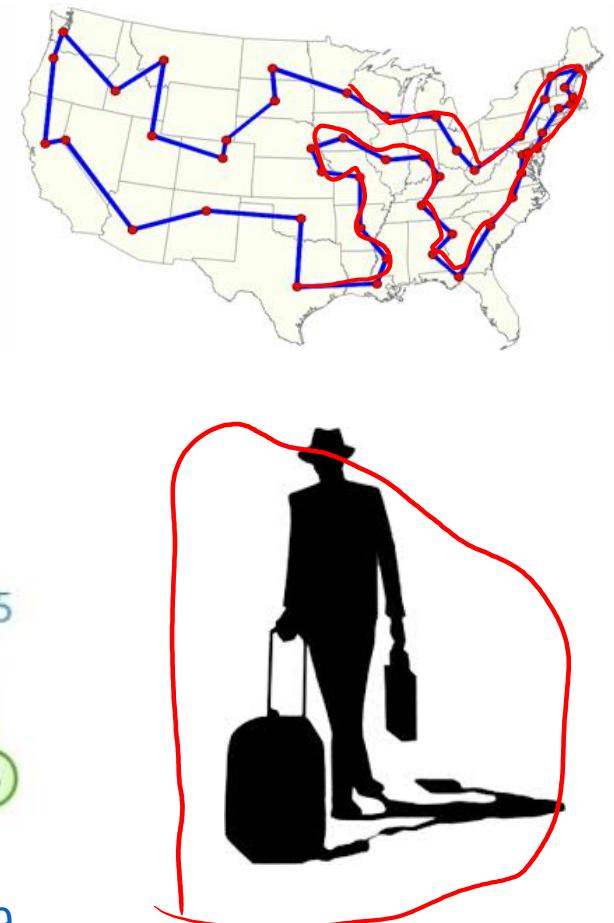
➤ Optimization problems are everywhere...

Travelling Salesman Problem

- Many of them are notoriously hard problems.

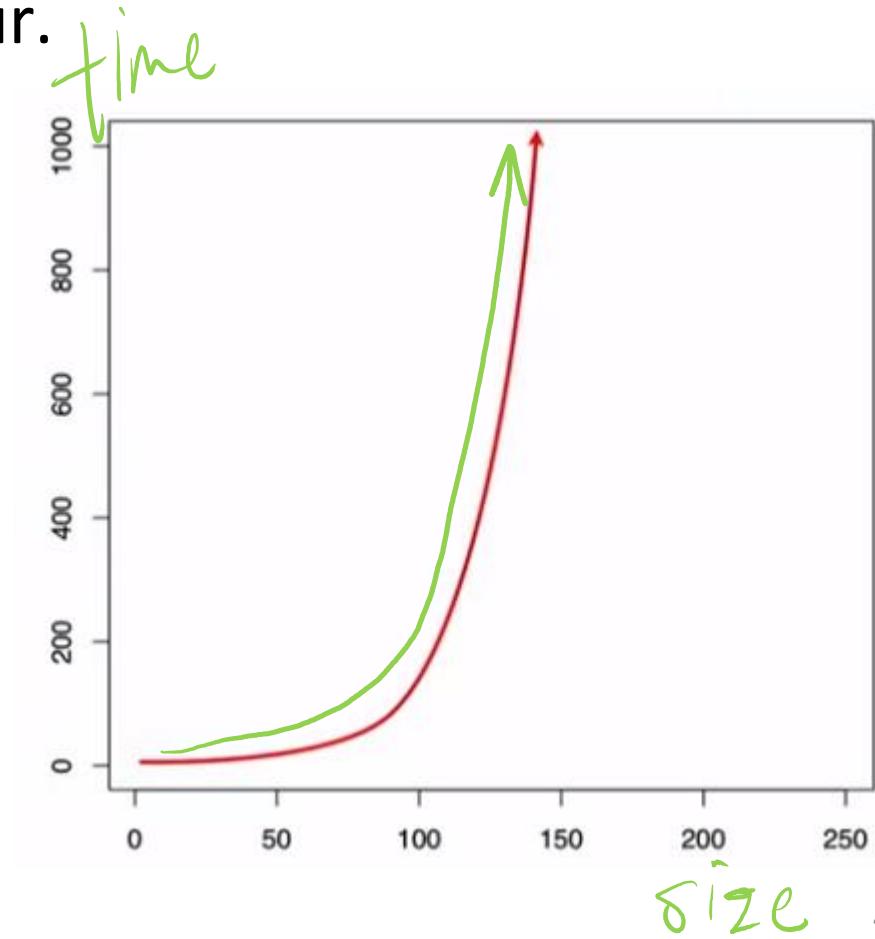
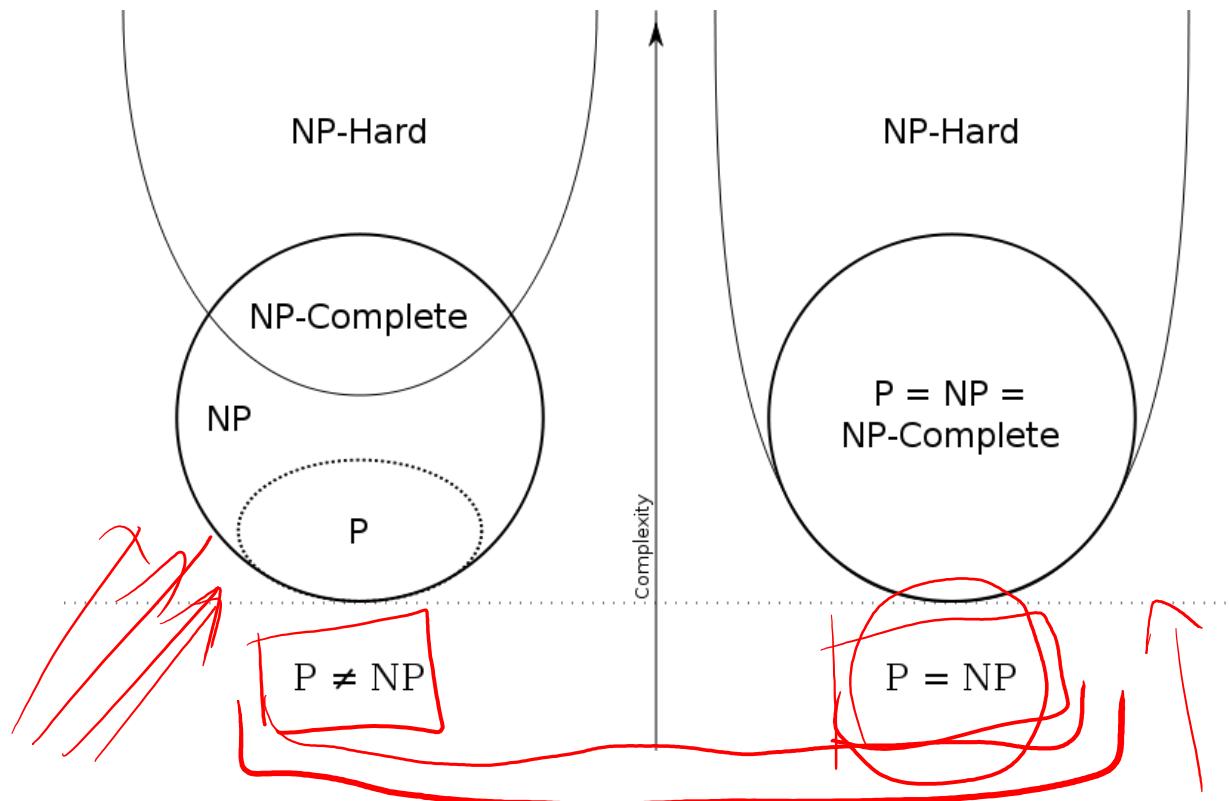


Source: [Park et al., 2019](#)



Np-Complete Problem

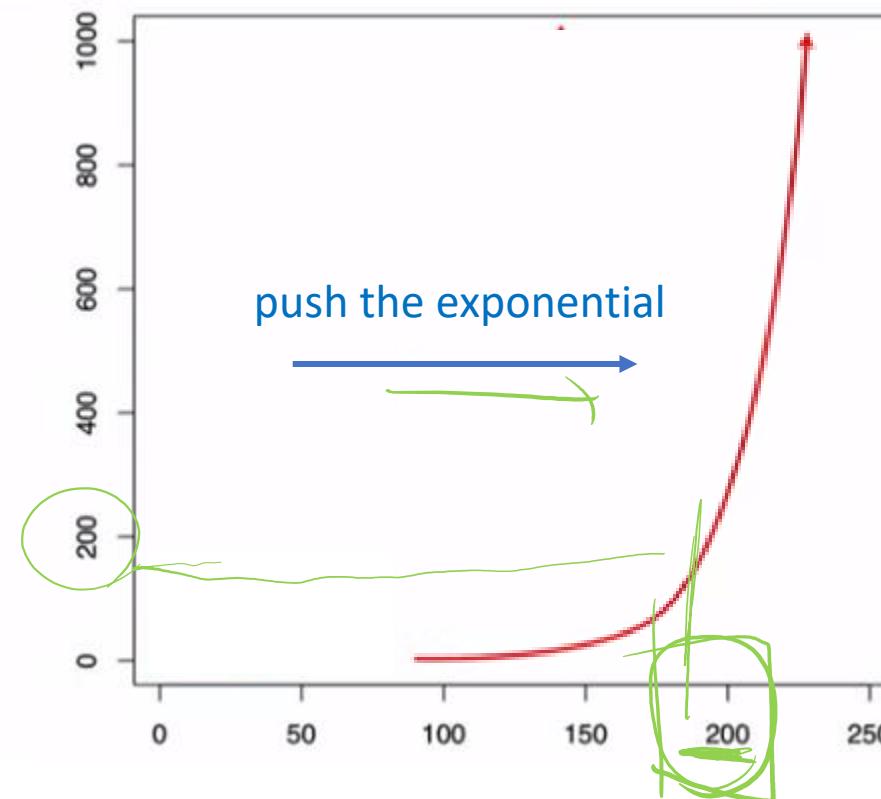
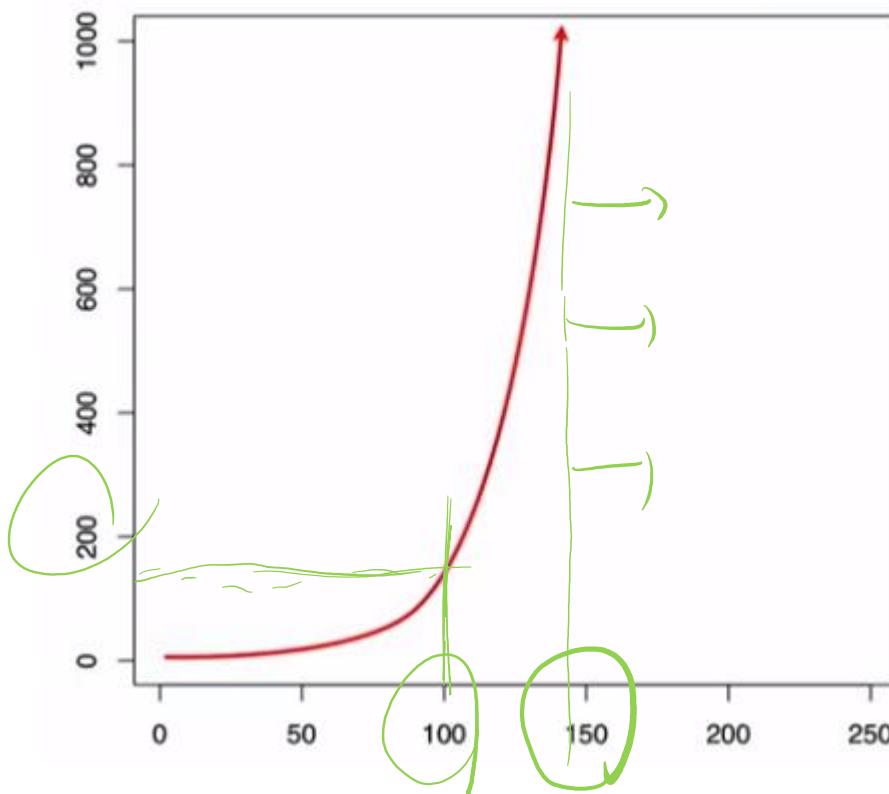
- If we have a solution we can evaluate it quickly, but finding a solution is not trivial and has exponential behaviour.



size

Difficult to solve

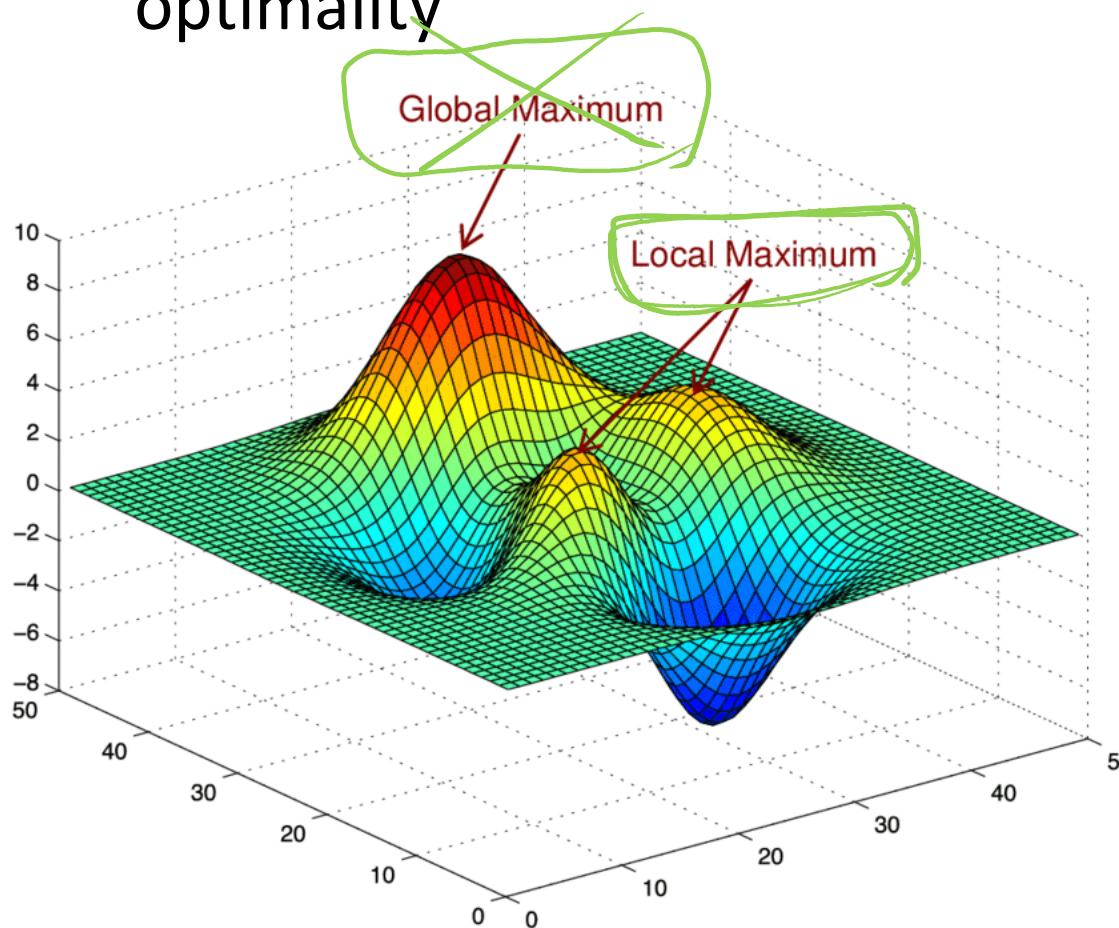
- We may be able to solve the problem for a small range of inputs, but exponential behavior can quickly become impractical...



Sometimes we can adjust the problem to make it feasible for a practical range of inputs.

Difficult to solve

- Sometimes it is so hard that we cannot solve the problem to global optimality



- We still have to solve the problem in some way...
- ... what we can do is relaxing the definition of “solving a problem”
- More precisely we don’t focus on finding a globally “optimal” solution.

Example: Knapsack Problem

- There are several approaches to solving these problems. Let us demonstrate with a popular Knapsack problem.



- Which treasure do we select?

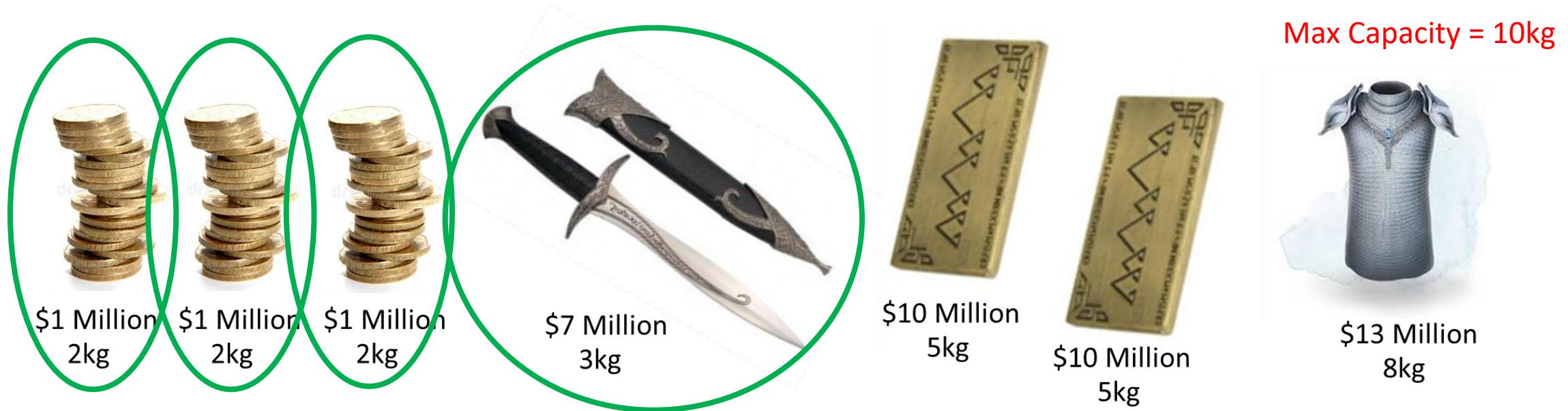
Knapsack Problem: Attempt 1



- Order the treasure by value and stuff your bag in order from most expensive to least.

\$14 Million
Greedy Algorithm!

Knapsack Problem: Attempt 2

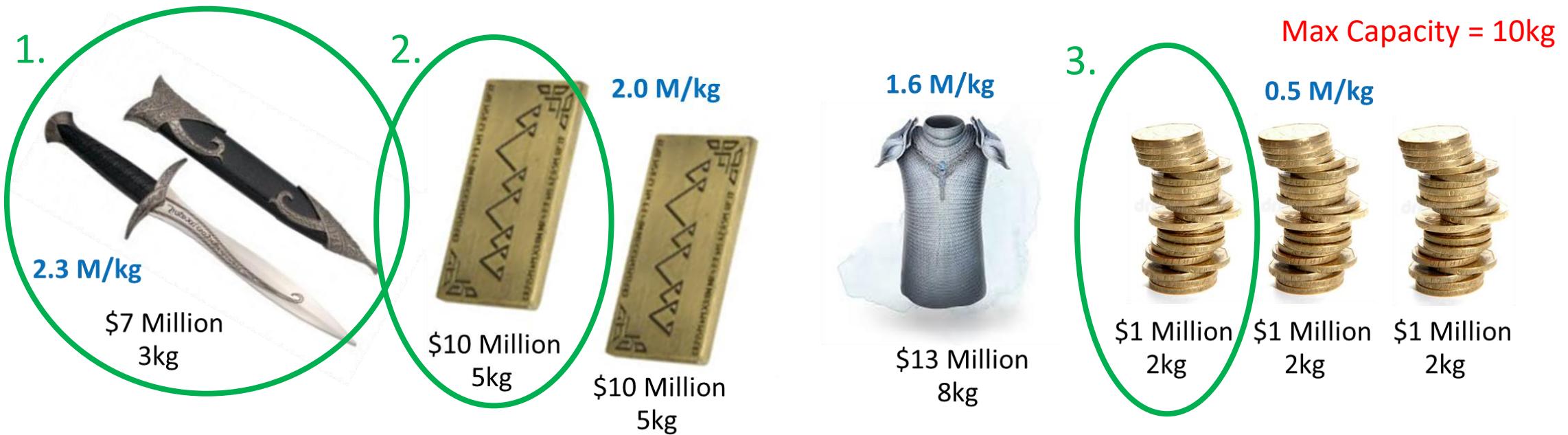


- Another intuition might be to try in the opposite order and start with the smallest weighing items hoping that you can pack more.

\$10 Million

Greedy Algorithm!

Knapsack Problem: Attempt 3



➤ You could consider the true problem we're trying to maximize (**value per weight**).

\$18 Million
Greedy Algorithm!

Knapsack Problem: Optimal Solution?



Example: Set Cover Problem

- Set cover is a classical problem in combinatorics!
 - Given a universe U of n elements ($U = \{1,2,\dots,n\}$), a collection of subsets $S = \{S_1, \dots, S_k\}$ of U , what is the smallest/cheapest sub-collection of S whose union equals the universe U .
 - A Cover is a subfamily C of sets (from S) for which the union is U
-
- For example:
- Consider a universe $U = \{1,2,3,4,5\}$ and the collection of sets $S = \{\{1,2,3\}, \{2,4\}, \{3,4\}, \{4,5\}\}$. Identify the smallest sub-collection of S whose union equals the universe.

Example with Cost Associations

Q: Consider this instance:

- $U = \{1, 2, 3\}$,
- $S = \{S_1, S_2, S_3\}$ with $S_1 = \{1, 2\}$, $S_2 = \{2, 3\}$, $S_3 = \{1, 2, 3\}$
- and cost $c(S_1) = 10$, $c(S_2) = 50$, and $c(S_3) = 100$.

Given that these collections cover U : $\{S_1, S_2\}$, $\{S_3\}$, $\{S_1, S_3\}$, $\{S_2, S_3\}$, $\{S_1, S_2, S_3\}$.

What is the cheapest combination?

A: The cheapest one is $\{S_1, S_2\}$ with a cost equal to 60.

More Formally

Problem 5.1 SET COVER

Instance. Universe U with n elements, collection $\mathcal{S} = \{S_1, \dots, S_k\}$, $S_i \subseteq U$, a cost function $c : \mathcal{S} \rightarrow \mathbb{R}$.

Task. Solve the problem

Minimize cost of sets (or # of sets, if costs are 1)

$$\text{minimize } \text{val}(x) = \sum_{S \in \mathcal{S}} c(S)x_S,$$

All elements are covered (at least once) subject to $\sum_{S: e \in S} x_S \geq 1 \quad e \in U,$
 $x_S \in \{0, 1\} \quad S \in \mathcal{S}.$

Variable indicating whether it's chosen or not

The Greedy Algorithm

- Iteratively pick the most cost-effective set and remove the covered elements, until all elements are covered.

Input. Universe U with n elements, collection $\mathcal{S} = \{S_1, \dots, S_k\}$, $S_i \subseteq U$, a cost function $c : \mathcal{S} \rightarrow \mathbb{R}$.

Output. Vector $x \in \{0, 1\}^k$

C -> sets of elements already covered, x -> vector of chosen sets

Step 1. $C = \emptyset$, $x = 0$.

Step 2. While $C \neq U$ do the following: ***Until we have all elements of U covered***

- Find the most cost-effective set in the current iteration, say S .
- Set $x_S = 1$ and for each $e \in S - C$ set $\text{price}(e) = c(S)/|S - C|$.
- $C = C \cup S$.

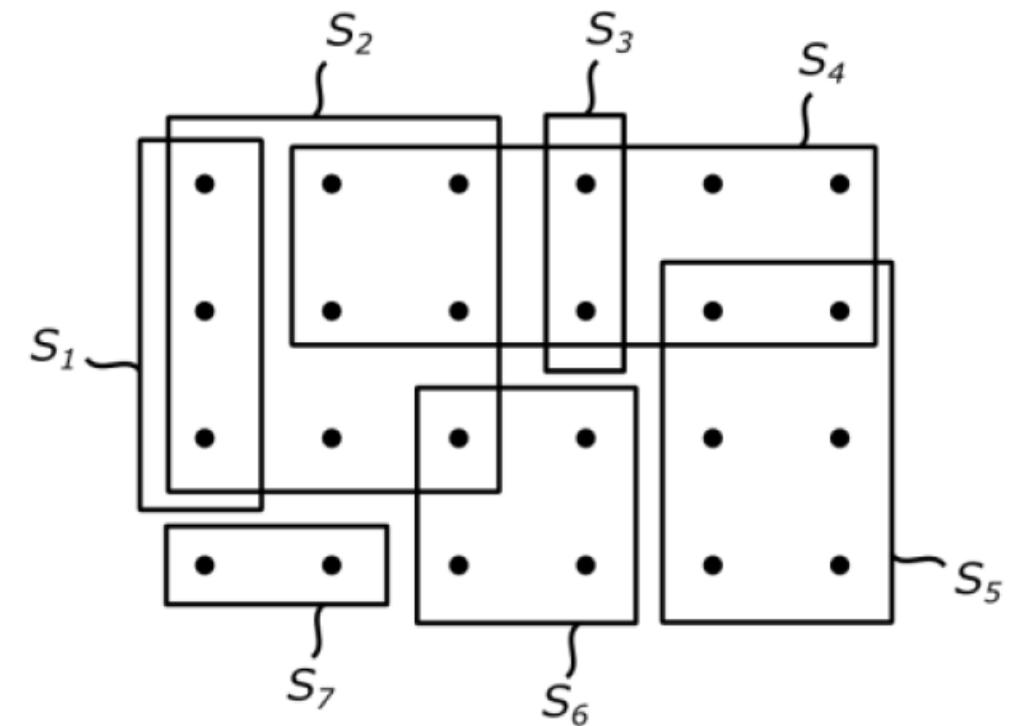
Cost of set / Elements not yet added

Cost-effectiveness of a set S – the average cost of covering new elements

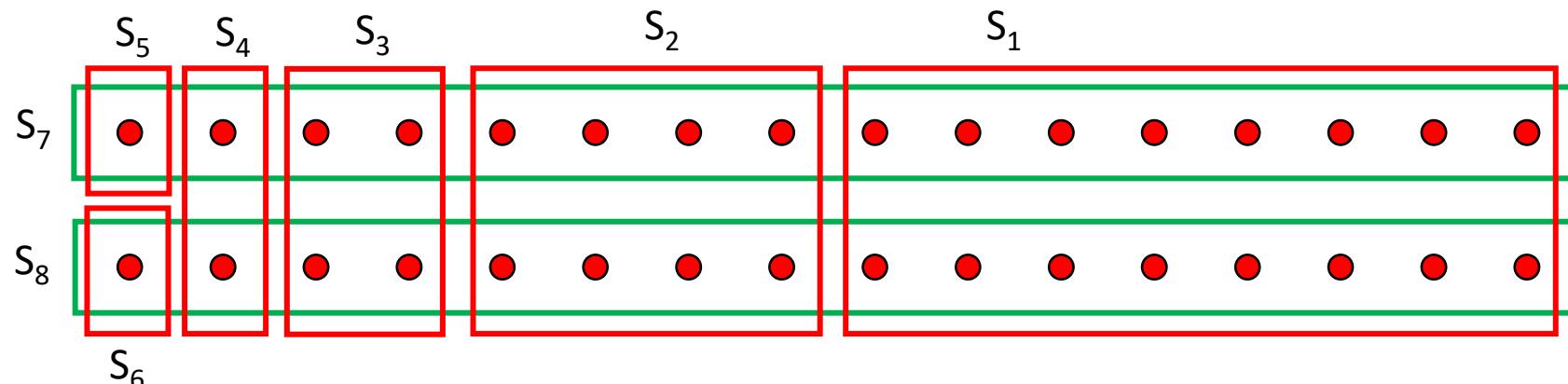
Step 3. Return x .

Example: Past Final Exam

The schematic to the right has sets S_1 , S_2 , S_3 , S_4 , S_5 , S_6 and S_7 . What sets, and in what order, would a greedy algorithm select to cover the universe (i.e., cover each point) if all sets are weighted equally?



Approximation factor



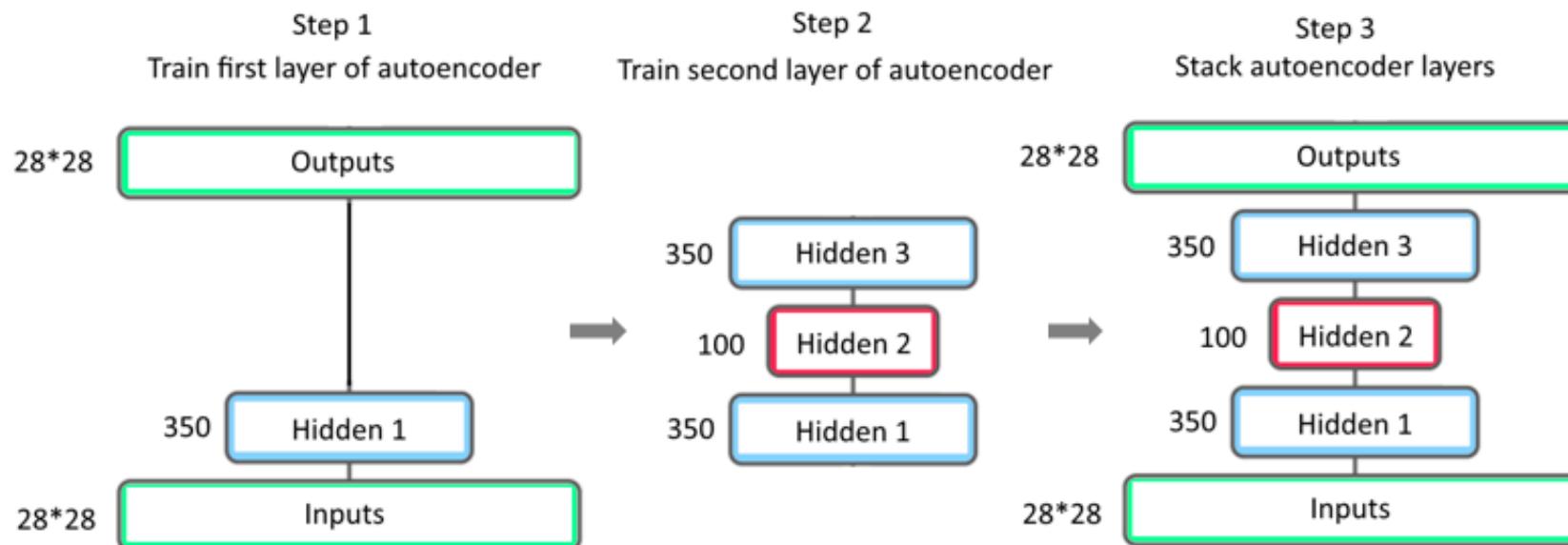
➤ Optimal is 2 sets, Greedy Algorithm finds 6 (off by a factor of 3)

Key Takeaways

- Optimization isn't only continuous (gradient descent), but can also be discrete -> requires a different way of thinking
- Greedy algorithms are sometimes useful methods for obtaining a “good” heuristic solution.

Link to Machine Learning

- We used a Greedy Discrete Optimization Algorithm in Project 1 to select the top features.
- Some deep neural networks can be trained using a greedy approach.
- For example, a Deep Autoencoder:



The End

or Just the Beginning...

- The MEng in MIE with Emphasis in Analytics builds on the foundations covered in APS1070.
- Courses to consider:
 - MIE1626 – Data Science Methods and Quantitative Analysis
 - MIE1517 – Introduction to Deep Learning
 - MIE1624 – Introduction to Data Science and Analytics
 - ECE1513 – Introduction to Machine Learning
 - MIE1628 – Big Data Science
 - APS1080 – Introduction to Reinforcement Learning
 - and many more...

Next Time

- Please consider completing the course evaluation (by April 3rd)
- Week 11 Q/A Support on Thursday and Friday
 - Project 4 is due on April 1st
- Week 12 Lecture - Review
 - Discussing past final exam questions
- Final Assessment (Crowdmark)
 - April 12th at 9:00 to April 13th at 15:00

