APS1070

Foundations of Data Analytics and Machine Learning

Winter 2022

Week 11:

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

Sinisa Colic and Samin Aref

Slide Attribution

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

- Marc Deisenroth
- Pascal Van Hentenryck

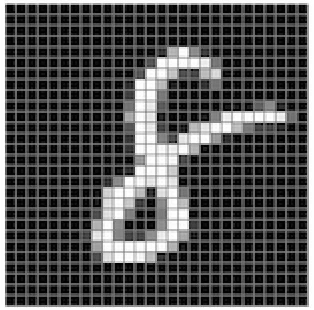
Last Time

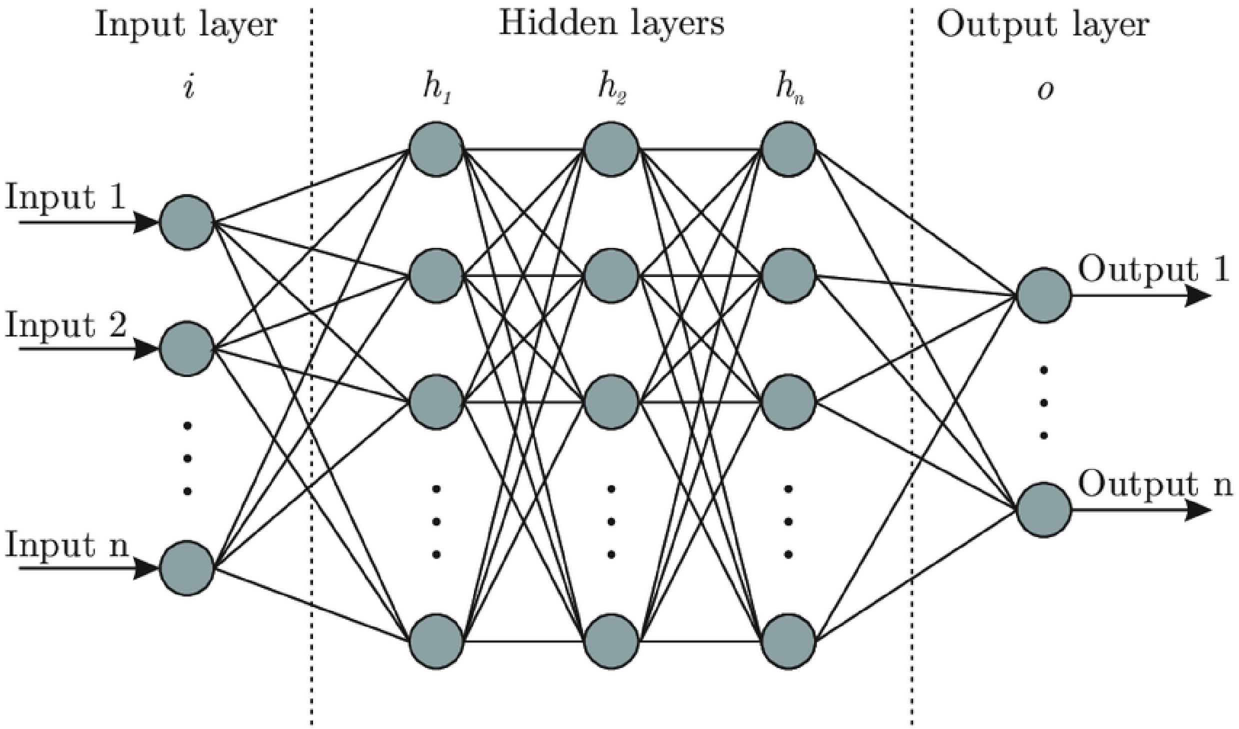
- Nonlinear Regression (and Classification)
  - Polynomial Regression
  - Regularization
  - Neural Networks

- 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.

Recap: Neural Networks

- Q: What makes neural networks so special?





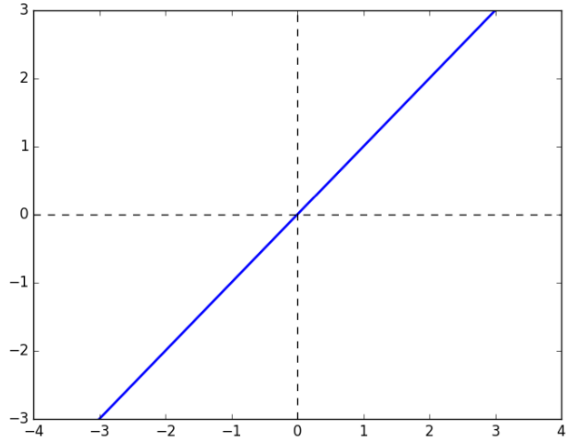
One-hot Output

[0,…0, 1, 0, …]

 # eight

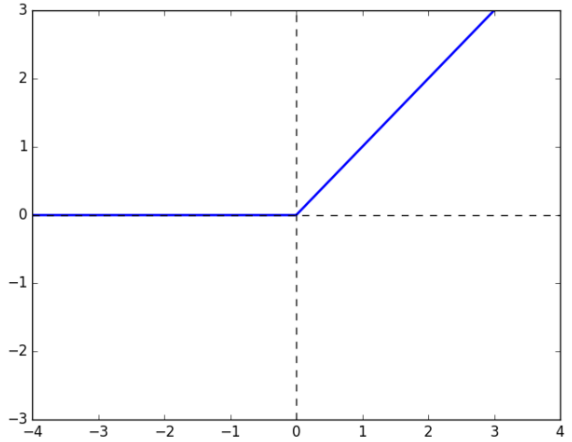
mathematically equivalent to a universal computer

Activation Functions



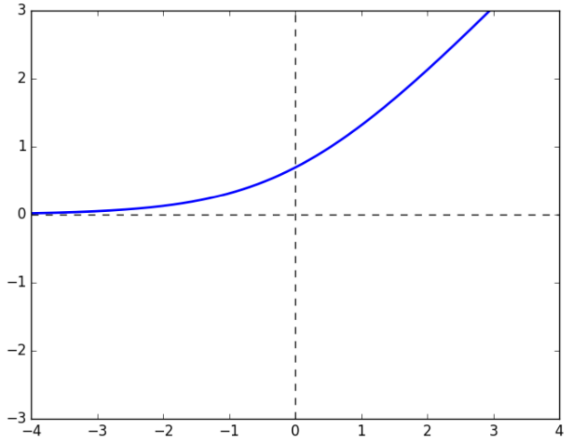
Linear

$$y = z$$



Rectified Linear Unit (ReLU)

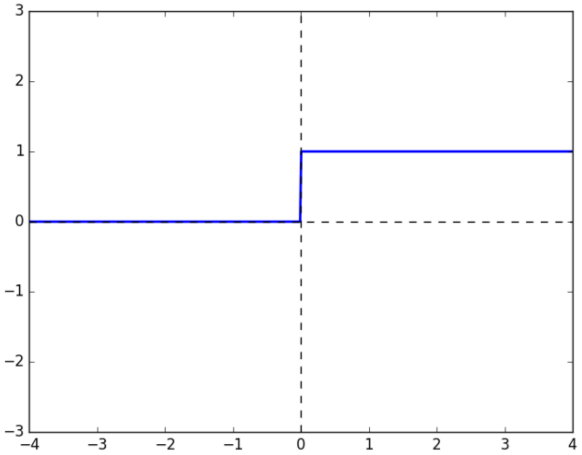
$$y = \max(0, z)$$



Soft ReLU

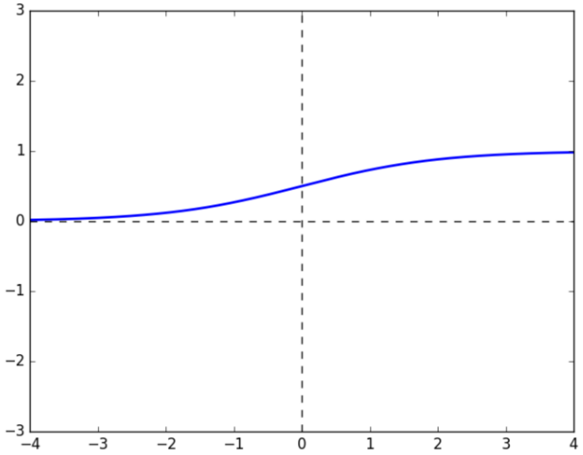
$$y = \log 1 + e^z$$

Activation Functions



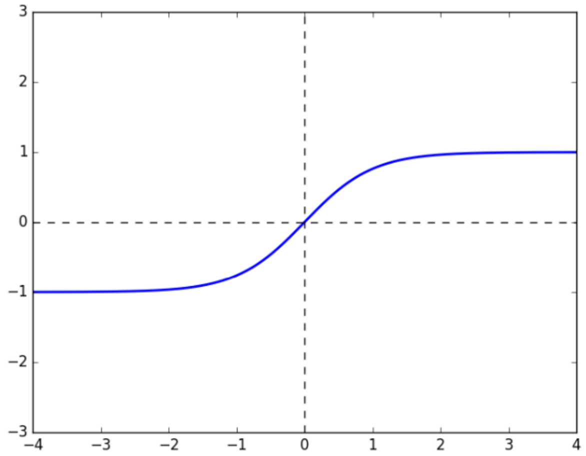
Hard Threshold

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



Logistic

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

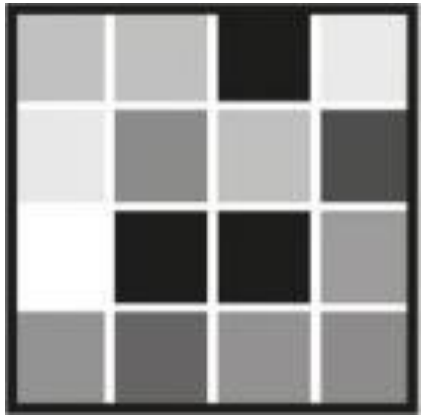


Hyperbolic Tangent (tanh)

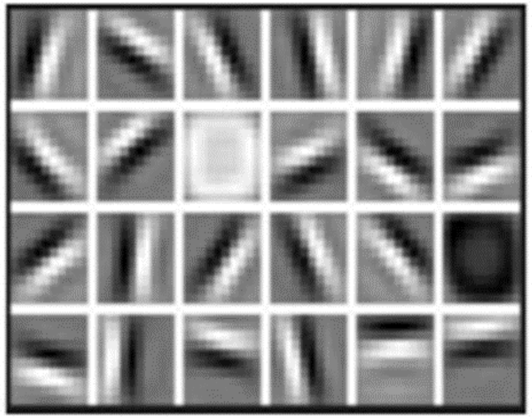
$$y = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Deep Learning

Deep-learning 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.

Agenda

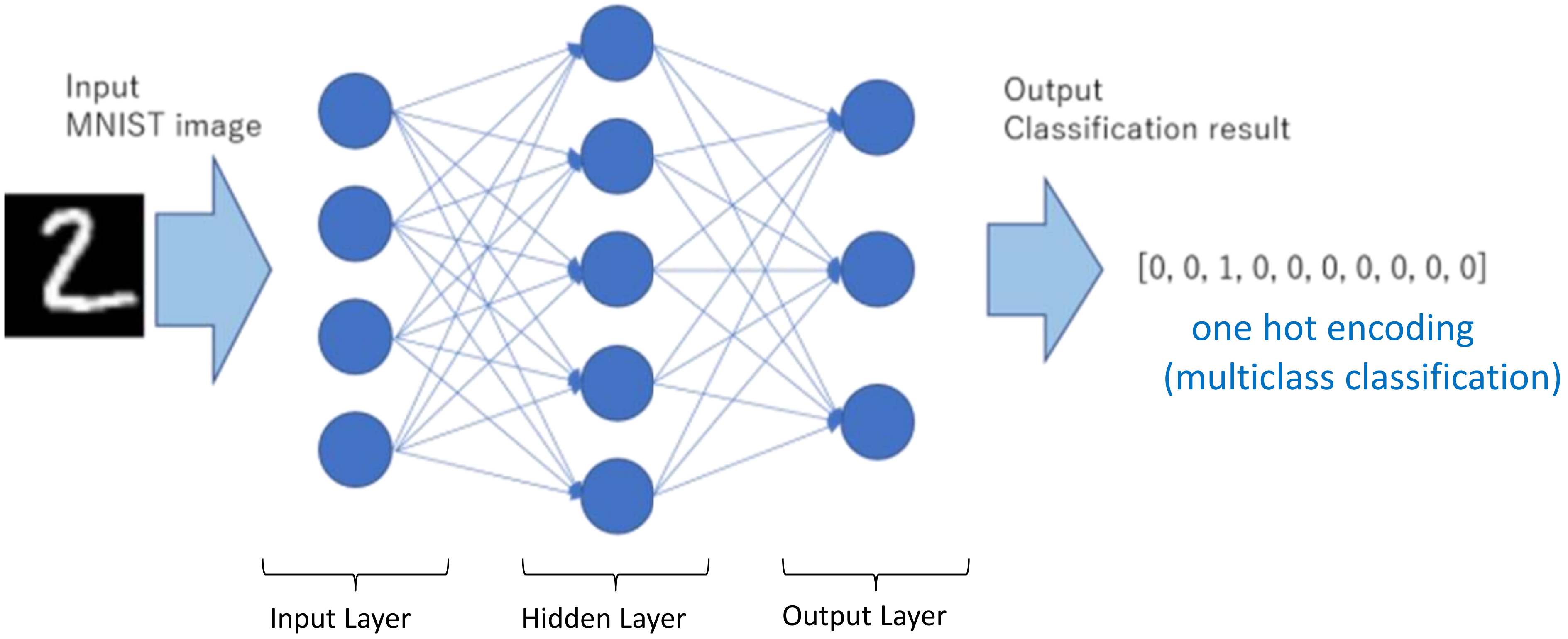
- Gradient Exercise
- Automatic Differentiation
- Deep Learning Architectures
- Transfer Learning

Theme:
Deep Learning

- More Concepts
  - Discrete Optimization

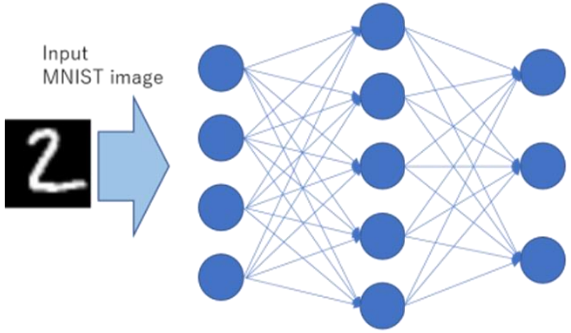
Take Home Exercise

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**.



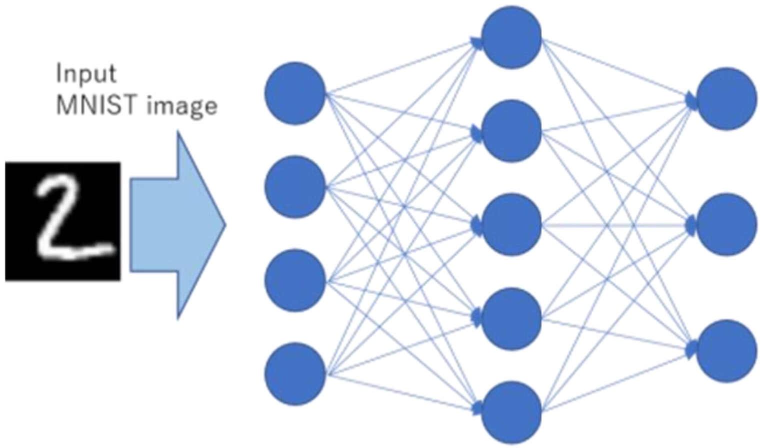
Gradients of a 2-layer Neural Network

A:

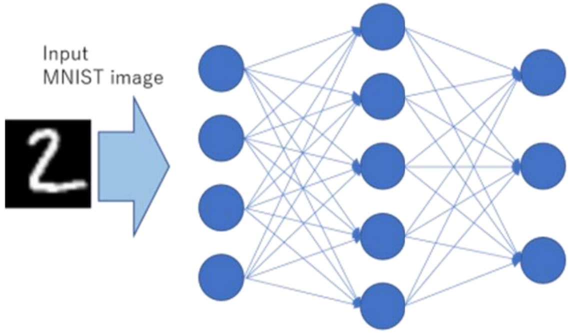


See NumPy Implementation

Let us take our computed gradients and implement them to solve a simple nonlinearly separable problem.



Verification of Gradients



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

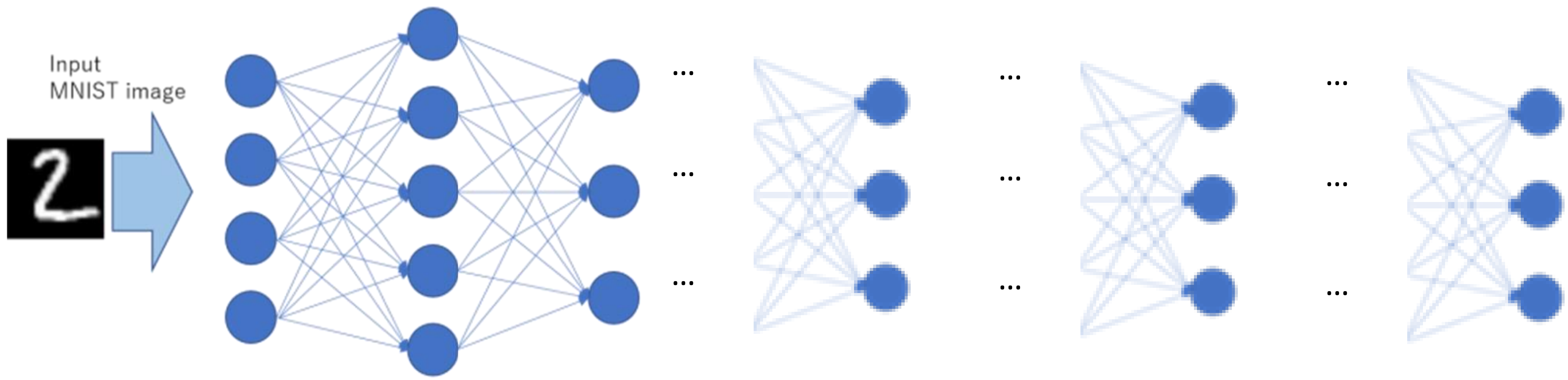
A: We can use a **numerical approach** to compute the gradient and compare to the analytically computed ones.

$$\frac{\partial f}{\partial x_i} = \lim_{h \to 0} \frac{f(x_1, \ldots, x_{i-1}, \boxed{x_i + h}, x_{i+1}, \ldots, x_N) - f(\boldsymbol{x})}{h}$$

see sample code with implementation

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:

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

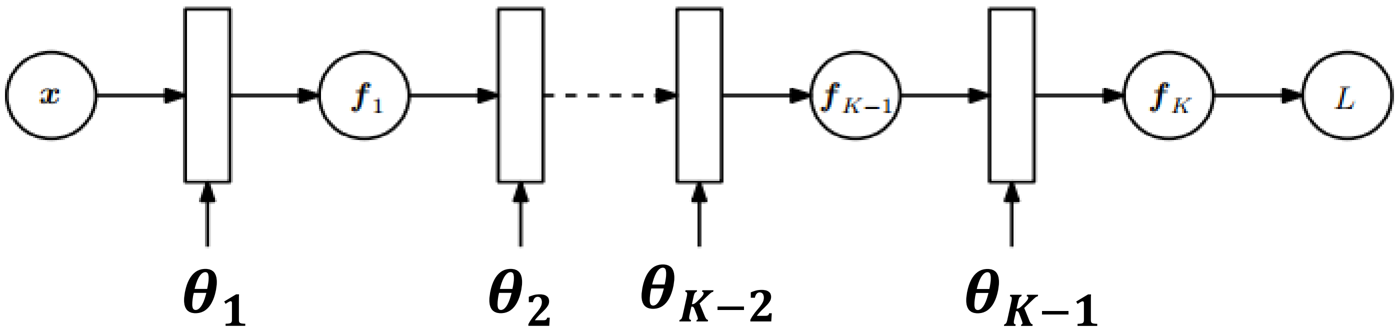
- Application of chain rule yields the following gradient:

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

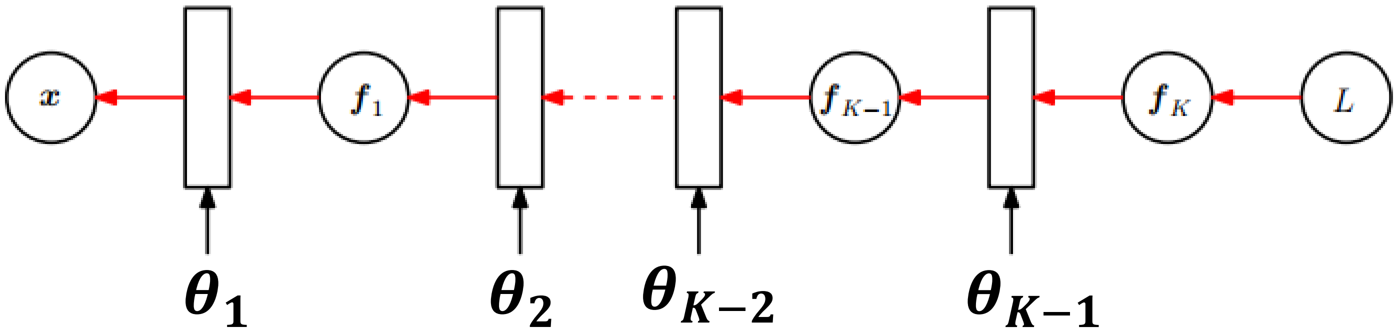
- Writing out the gradient in this explicit way is often impractical and could be significantly more expensive than computing the function.

Backpropagation

Forward Pass



Backward Pass



$$\frac{\partial L}{\partial \boldsymbol{\theta}_{K-1}} = \frac{\partial L}{\partial \boldsymbol{f}_K} \frac{\partial \boldsymbol{f}_K}{\partial \boldsymbol{\theta}_{K-1}}$$

$$\frac{\partial L}{\partial \boldsymbol{\theta}_{K-2}} = \frac{\partial L}{\partial \boldsymbol{f}_K} \boxed{\frac{\partial \boldsymbol{f}_K}{\partial \boldsymbol{f}_{K-1}} \frac{\partial \boldsymbol{f}_{K-1}}{\partial \boldsymbol{\theta}_{K-2}}}$$

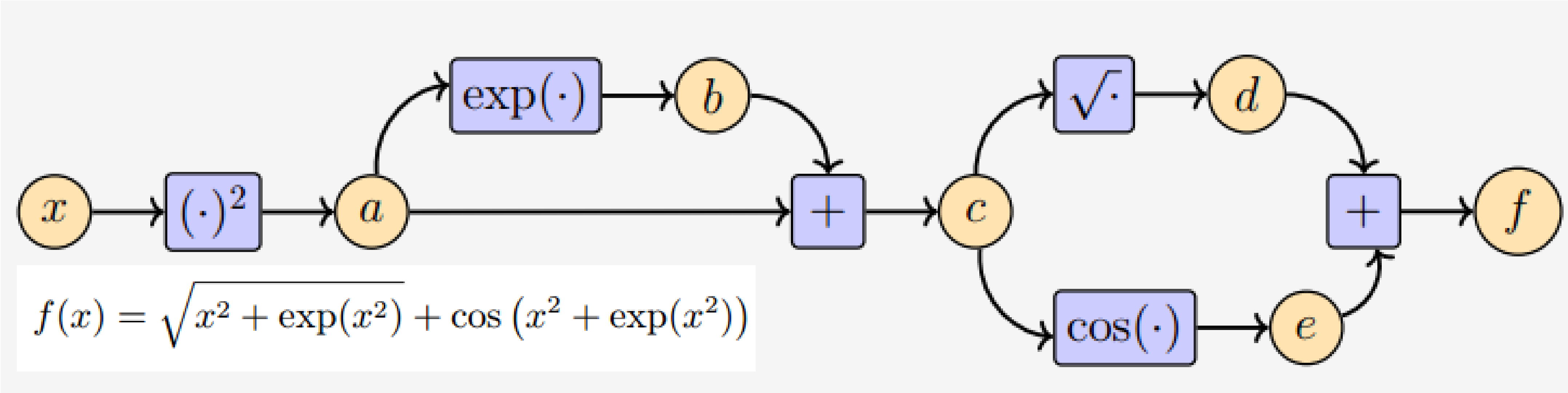
$$\frac{\partial L}{\partial \boldsymbol{\theta}_{K-3}} = \frac{\partial L}{\partial \boldsymbol{f}_K} \frac{\partial \boldsymbol{f}_K}{\partial \boldsymbol{f}_{K-1}} \boxed{\frac{\partial \boldsymbol{f}_{K-1}}{\partial \boldsymbol{f}_{K-2}} \frac{\partial \boldsymbol{f}_{K-2}}{\partial \boldsymbol{\theta}_{K-3}}}$$

$$\frac{\partial L}{\partial \boldsymbol{\theta}_i} = \frac{\partial L}{\partial \boldsymbol{f}_K} \frac{\partial \boldsymbol{f}_K}{\partial \boldsymbol{f}_{K-1}} \cdots \boxed{\frac{\partial \boldsymbol{f}_{i+2}}{\partial \boldsymbol{f}_{i+1}} \frac{\partial \boldsymbol{f}_{i+1}}{\partial \boldsymbol{\theta}_i}}$$

Only compute once!

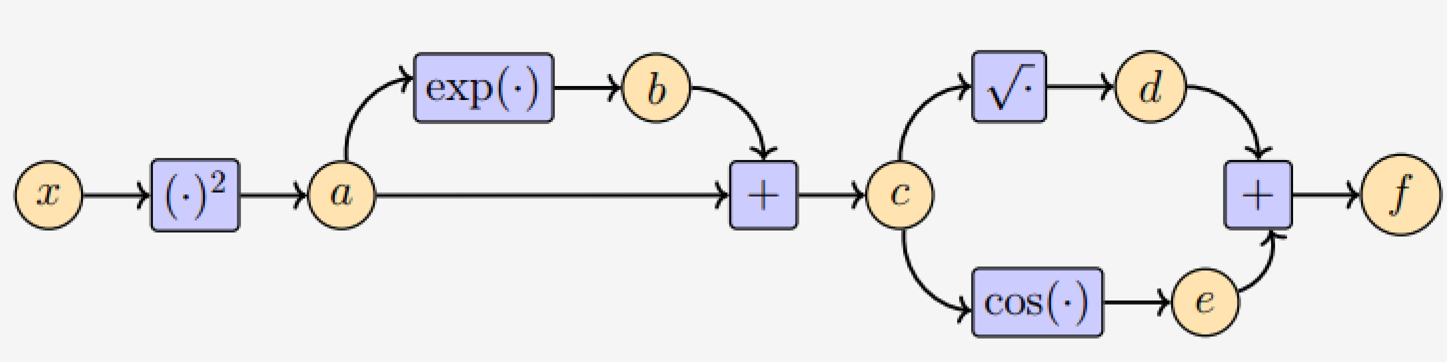
In order to train this network, we require the gradient of a loss function *L* with respect to all model parameters

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



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

1. Partial Derivatives

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

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

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

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

2. Chain Rule

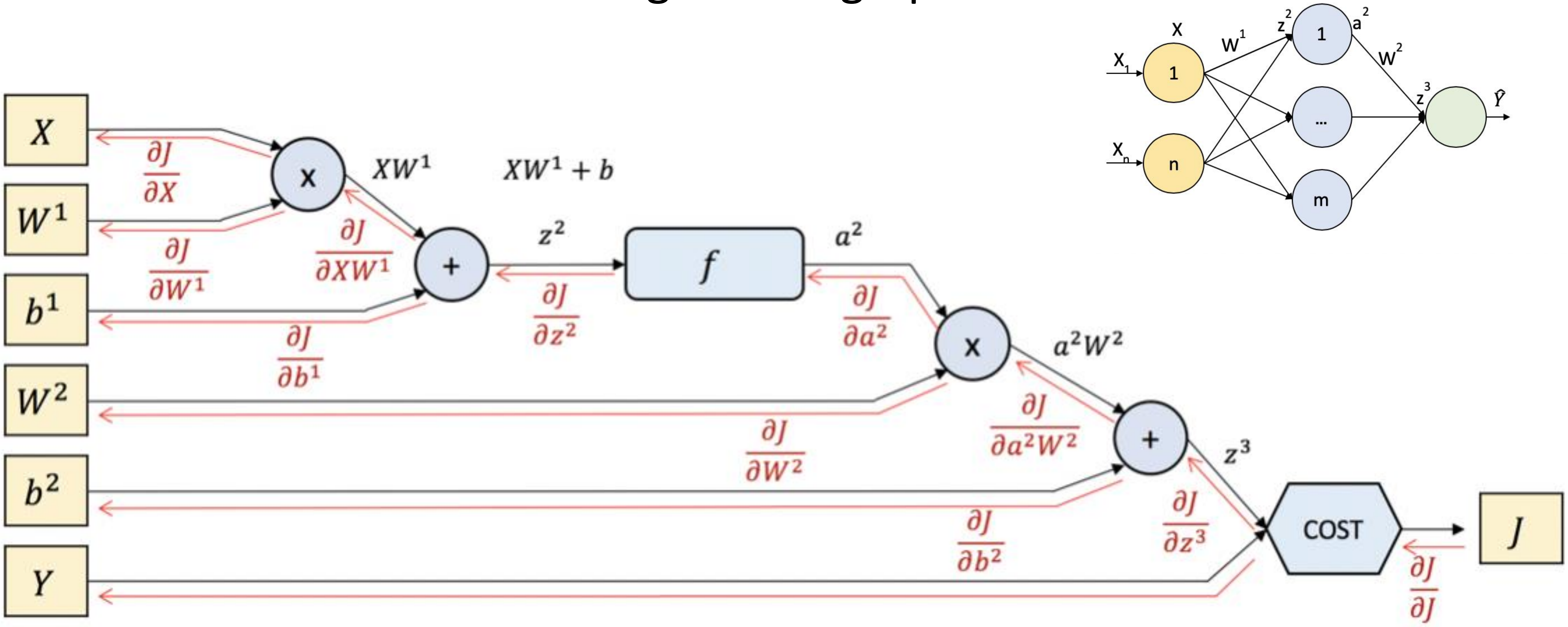
$$\begin{aligned}
\frac{\partial f}{\partial c} &= \frac{\partial f}{\partial d}\frac{\partial d}{\partial c} + \frac{\partial f}{\partial e}\frac{\partial e}{\partial c} \\
\frac{\partial f}{\partial b} &= \frac{\partial f}{\partial c}\frac{\partial c}{\partial b} \\
\frac{\partial f}{\partial a} &=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

3. Substitution

$$\begin{aligned}
\frac{\partial f}{\partial c} &= 1 \cdot \frac{1}{2\sqrt{c}} + 1 \cdot (-\sin(c)) \\
\frac{\partial f}{\partial b} &= \frac{\partial f}{\partial c} \cdot 1 \\
\frac{\partial f}{\partial a} &= \frac{\partial f}{\partial b}\exp(a) + \frac{\partial f}{\partial c} \cdot 1 \\
\frac{\partial f}{\partial x} &= \frac{\partial f}{\partial a} \cdot 2x\,.
\end{aligned}$$

Computation Graph

- Neural networks can be thought of as graphs.



Source: [Pablo Ruiz]

PyTorch

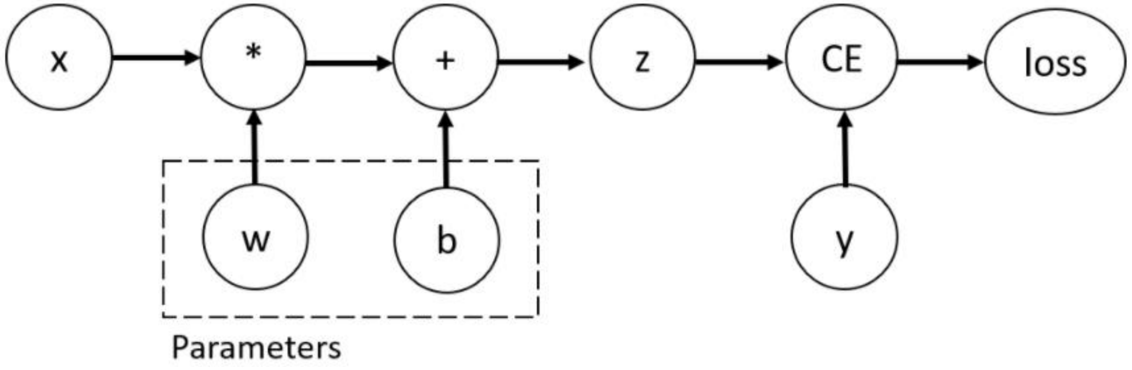
- One of several frameworks for handling gradients efficiently enabling GPUs for processing.
- Others include:
  - Tensorflow
  - Keras (built on Tensorflow)
  - Theano
  - and many more…

- 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?

Source: [PyTorch]

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
```

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

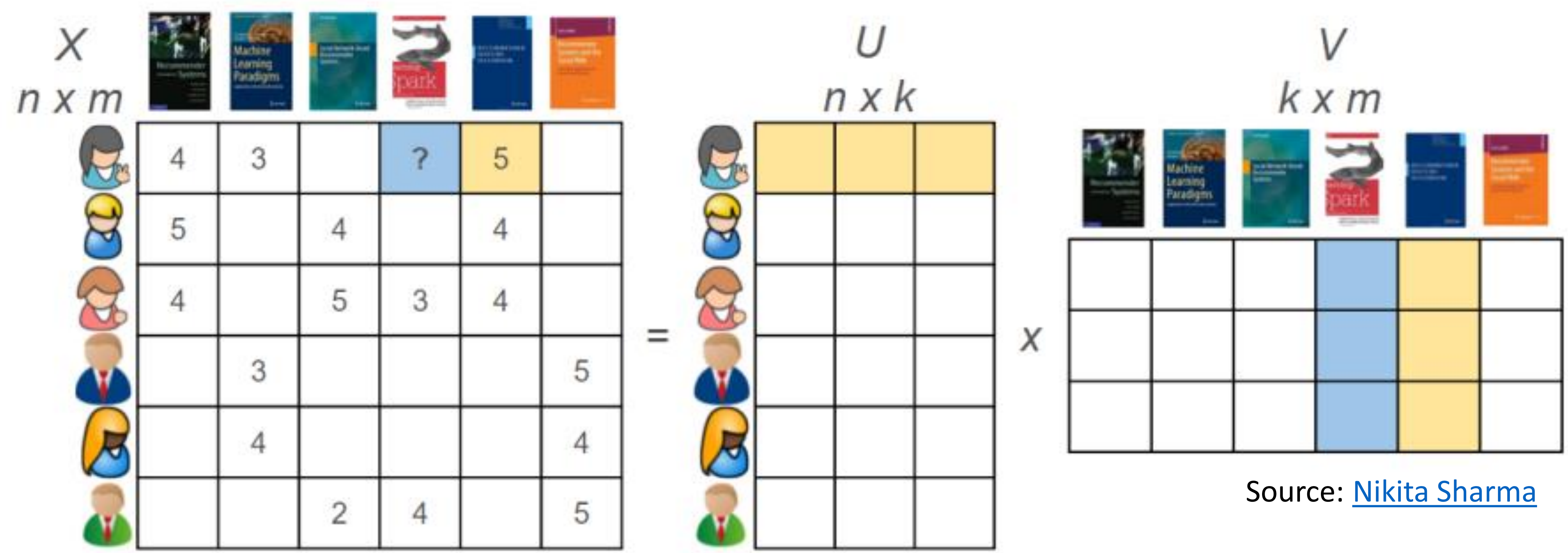
---- sample training code ----

```
#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()
```

PyTorch Sample Code

Example: Collaborative Filtering



Source: Nikita Sharma

Matrix decomposition using SVD does not work well with missing data.
Gradient descent can be used to learn U and V matrices to make movie recommendations.

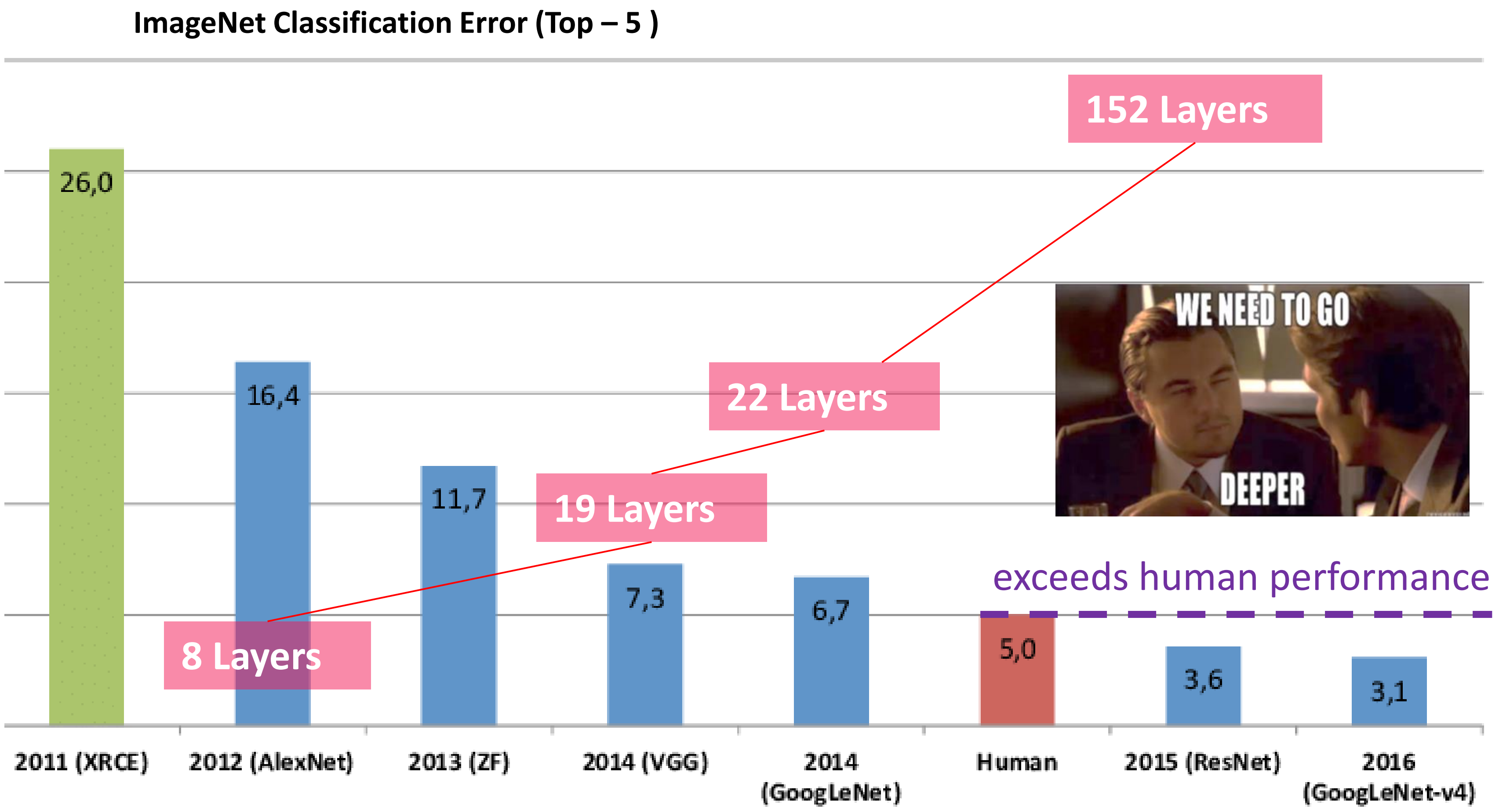
Deep Learning Architectures

Success of Deep Learning!

Large hand-labeled dataset

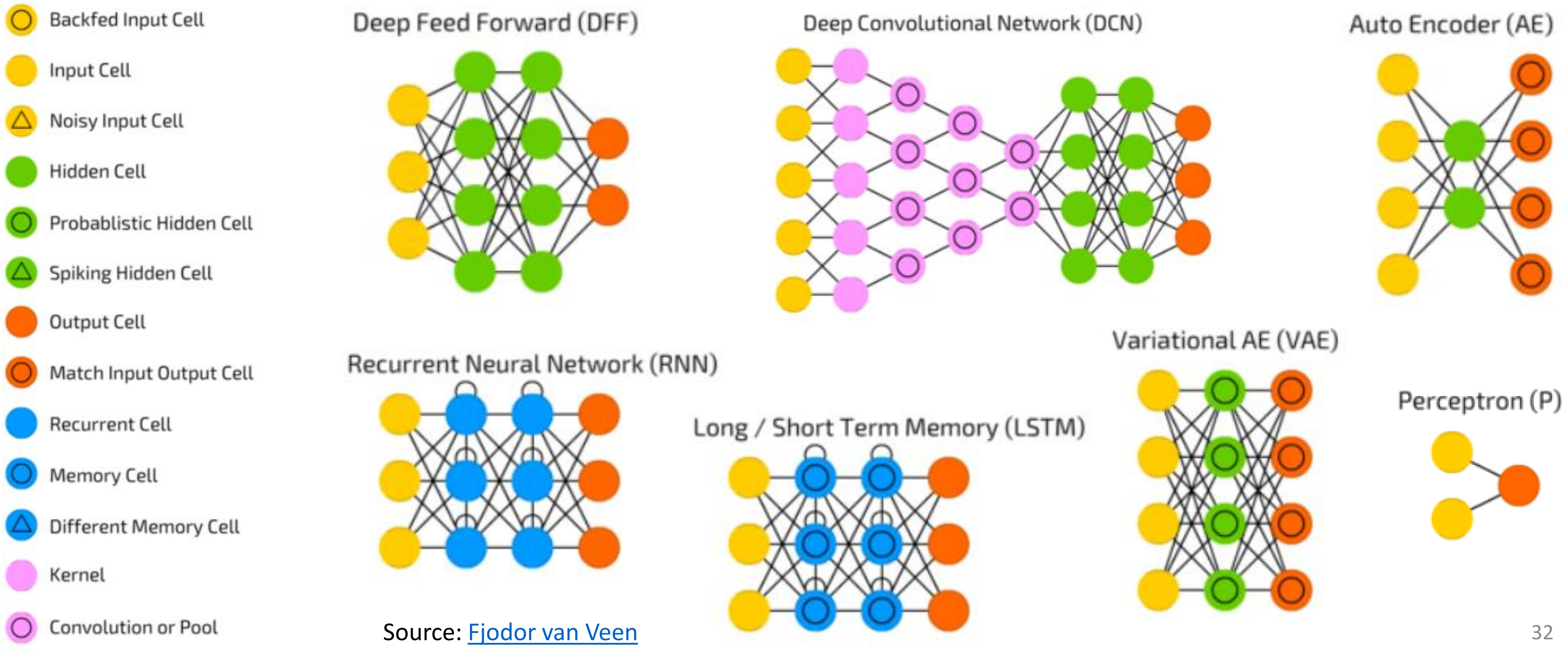
10,000,000 labeled images depicting 10,000+ object categories for training.

Algorithms assessed on unlabeled test images.



Graphical Representation

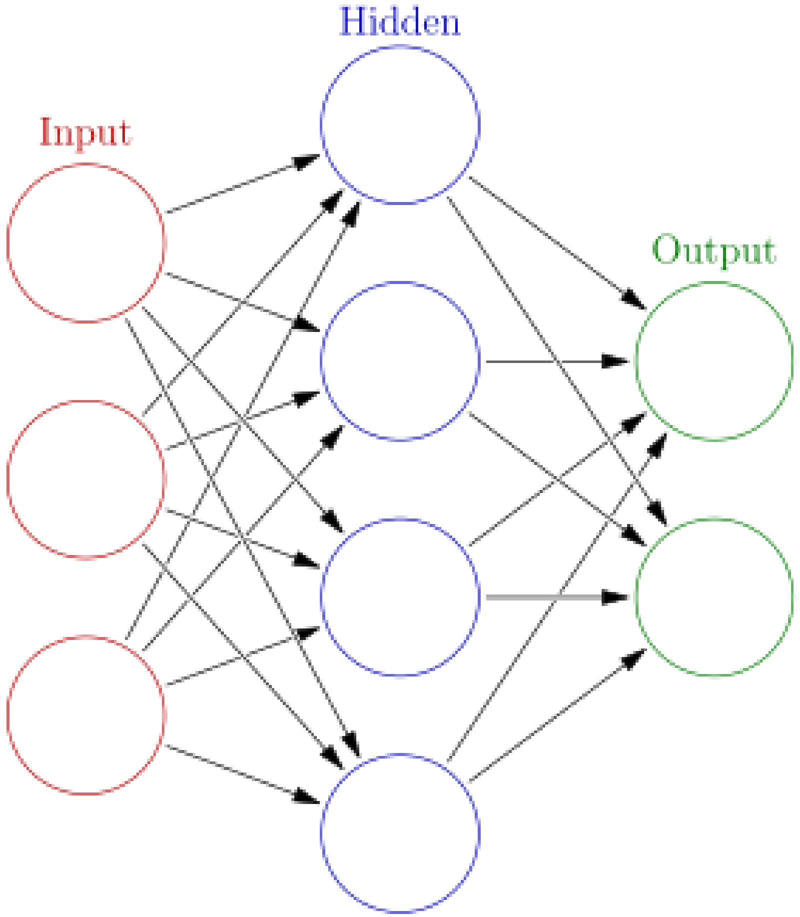
- Neural networks can be thought of as graphs.



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.

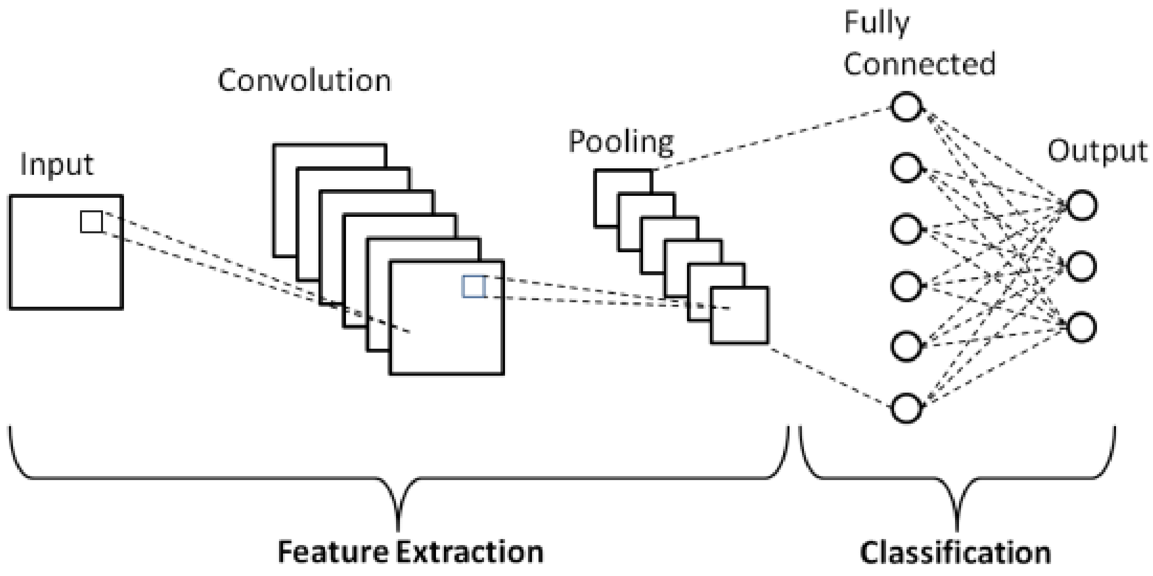


Convolutional Neural Networks

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

- Popular architectures include:
  - VCC18,
  - Inception (GoogLeNet)
  - ResNet

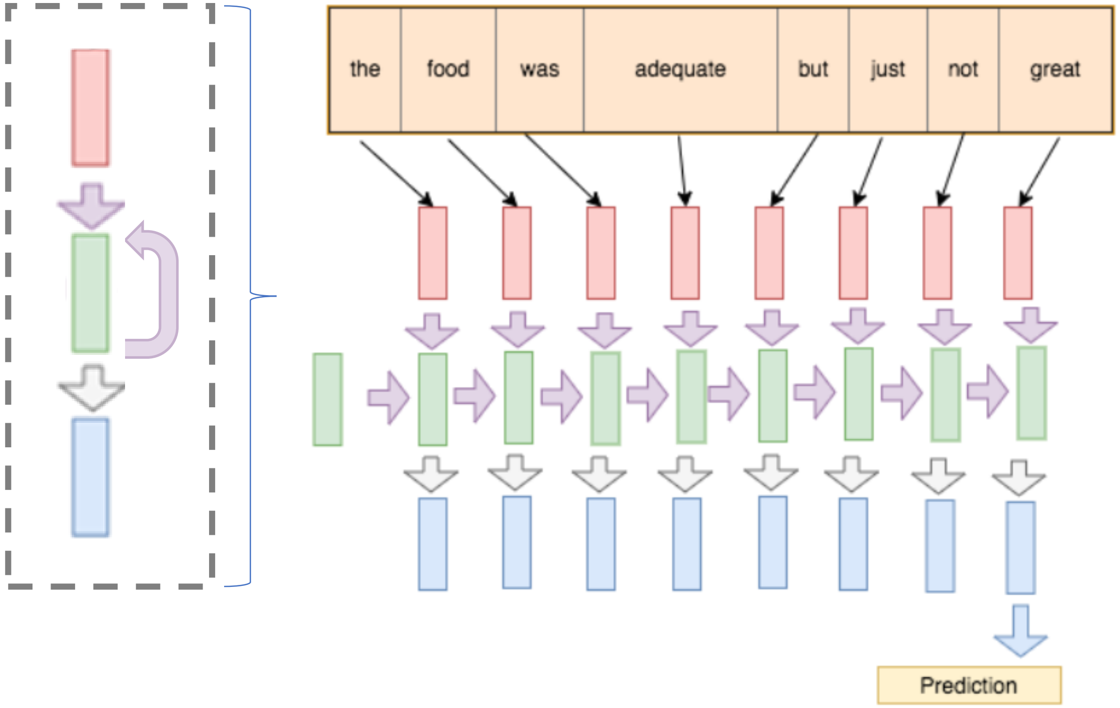
Example: Image Classification



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.

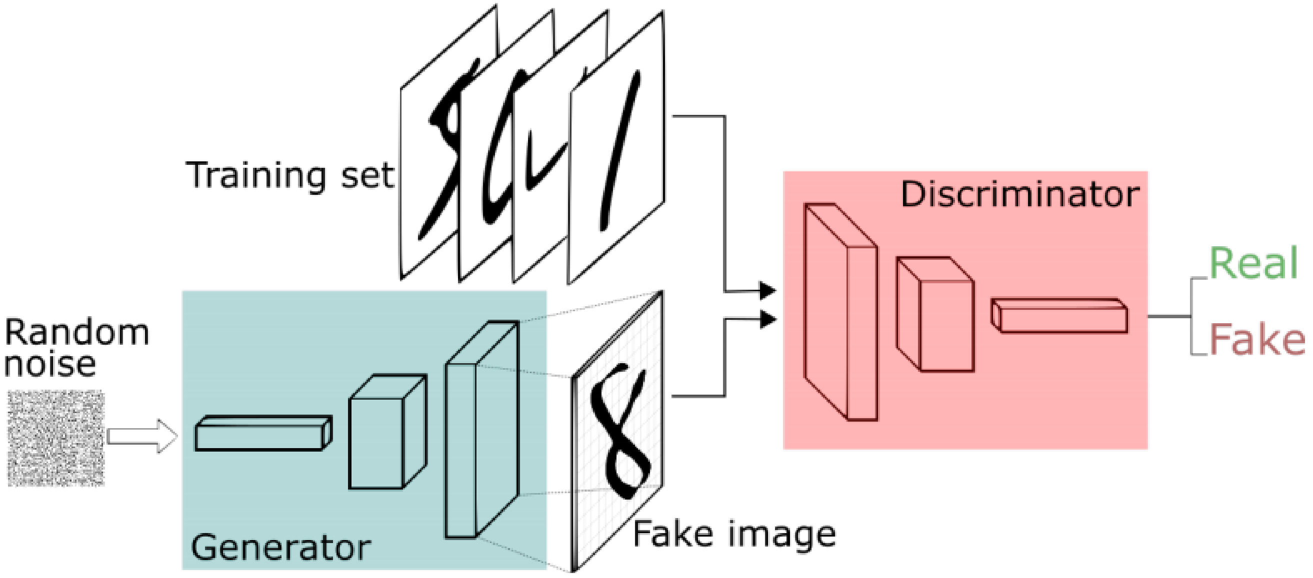
Example: Sentiment Analysis



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
  - …

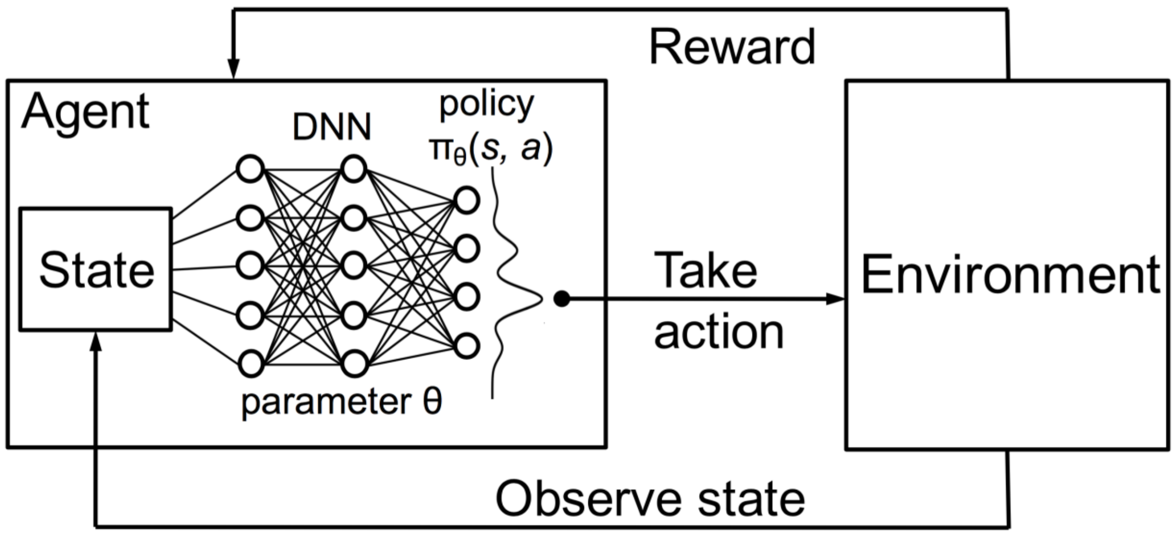
Example: Learn to generate digits



Analogy: Police vs Counterfeiters

Deep Reinforcement Learning

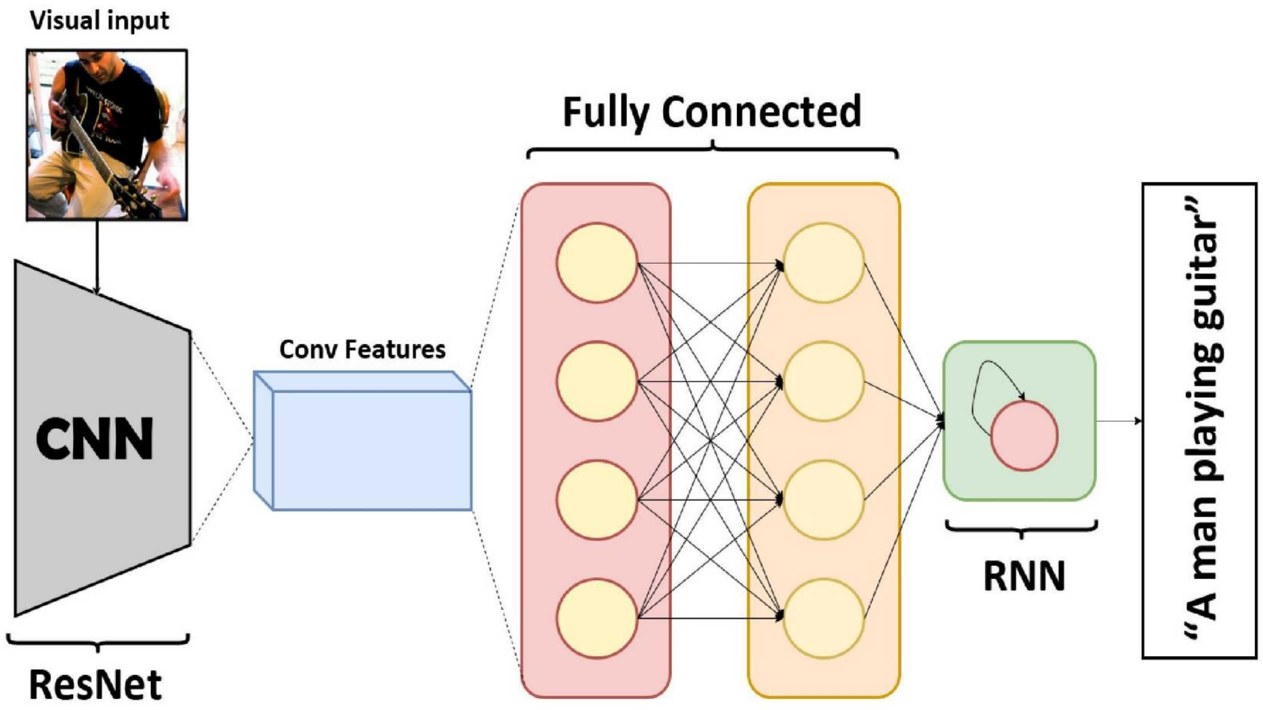
- **Key Ideas:**
  - handles real-world problems with asynchronous labels (aka rewards)
  - learns to generate a sequence of actions to maximize future rewards
- Many Success Stories:
  - exceed human performance on arcade games
  - AlphaGo champion at Go
  - Dota, Starcraft, etc.
  - …



Combining Neural Networks

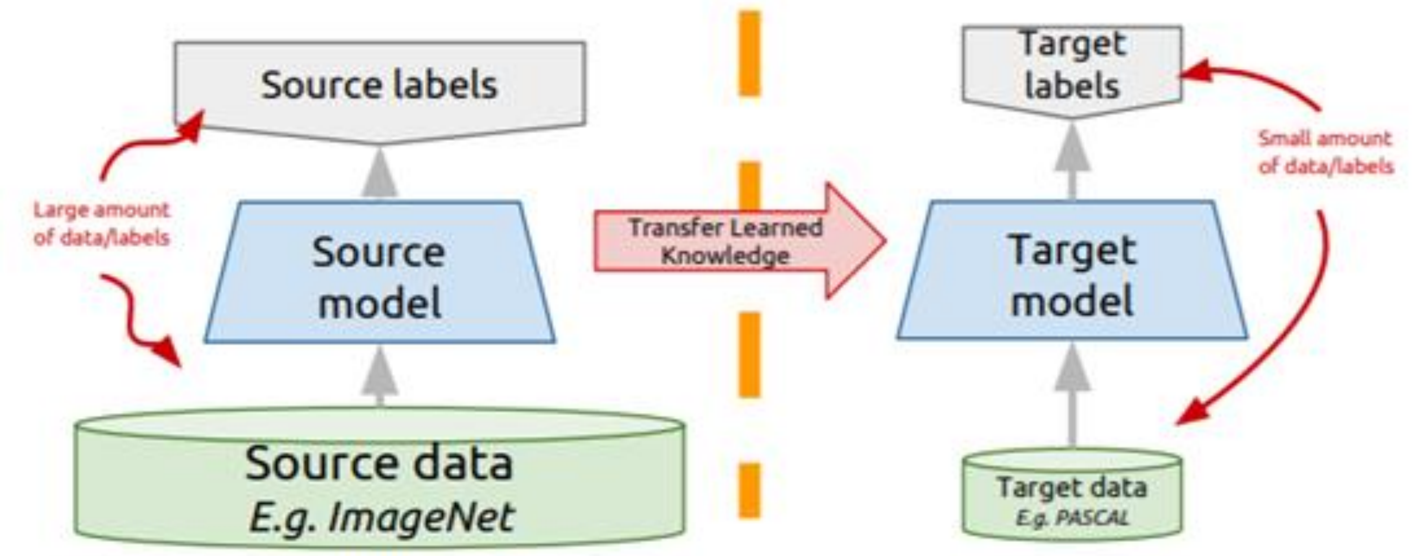
- Neural network architectures are often combined to transform data from input to output.
- Examples:
  - image captioning
  - video translation
  - sentiment analysis of videos

Example: Image Captioning



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)

Transformers

- Most of the success stories in transfer learning in the last decade have been confined to image processing tasks.
- Recent advances using transformer have brought transfer learning to 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

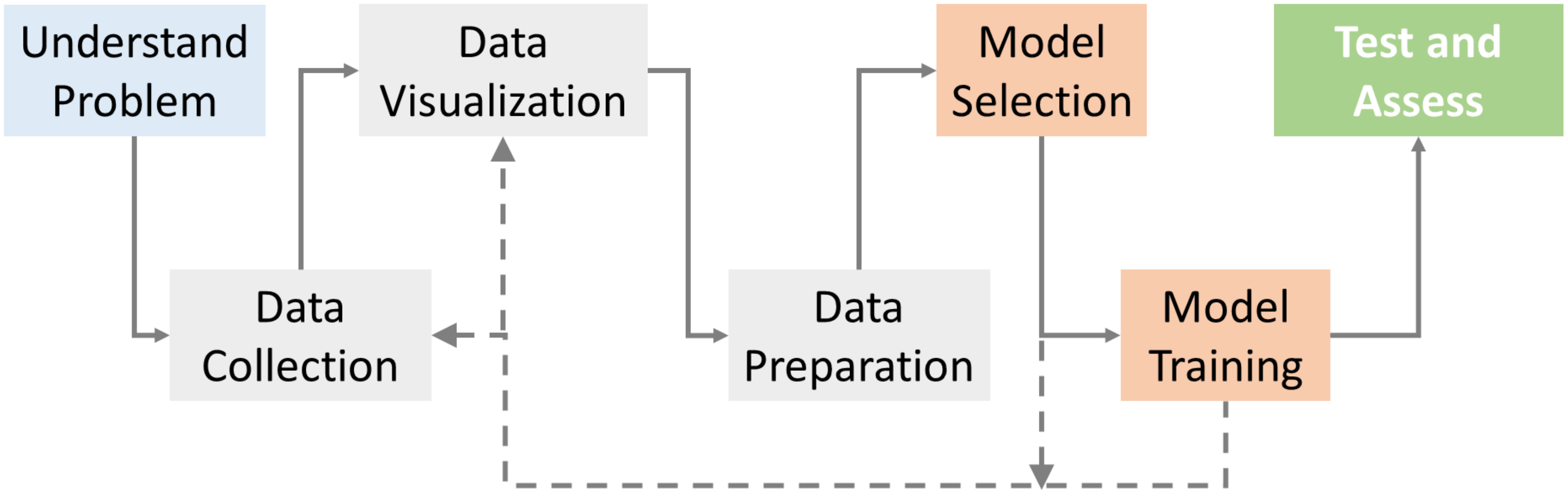
Deep Learning Summary

- 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.

End-to-End Machine Learning



More Concepts

Overview

- There are several concepts related to data science that deserve some consideration:
    - **Discrete Optimization**
    - Sampling Methods
    - Hypothesis Testing
    - Monte Carlo Sampling

Next Week

Discrete Optimization

Motivation

Logistics

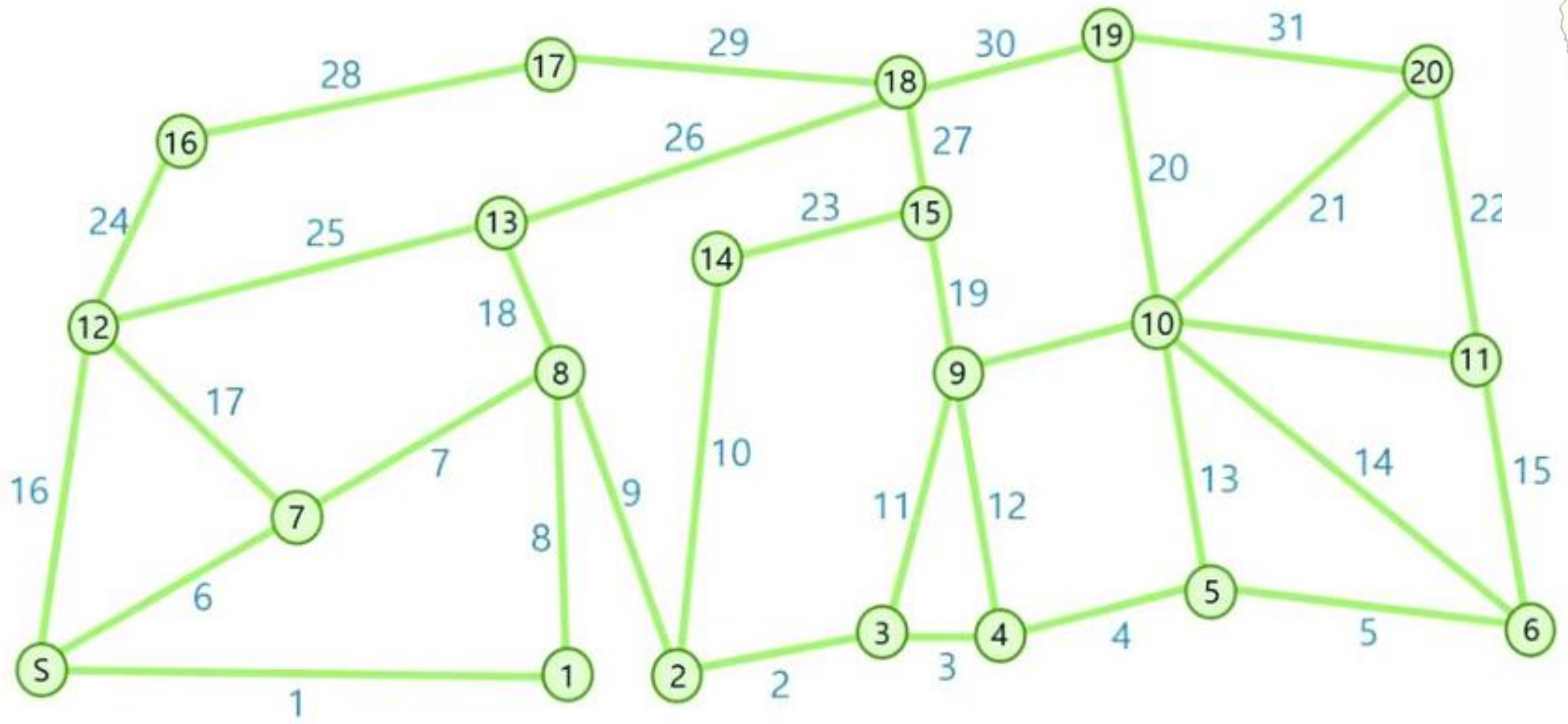
Energy

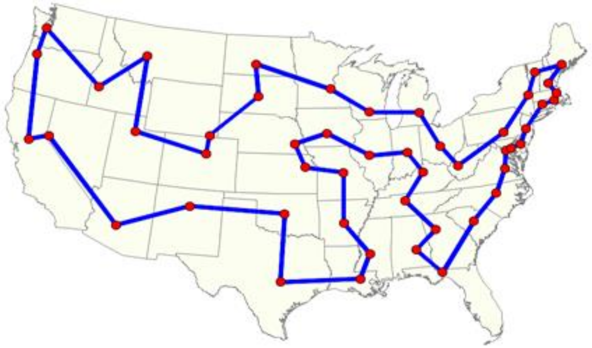
Scheduling

- Optimization problems are everywhere…

Travelling Salesman Problem

- Many of them are really hard problems.

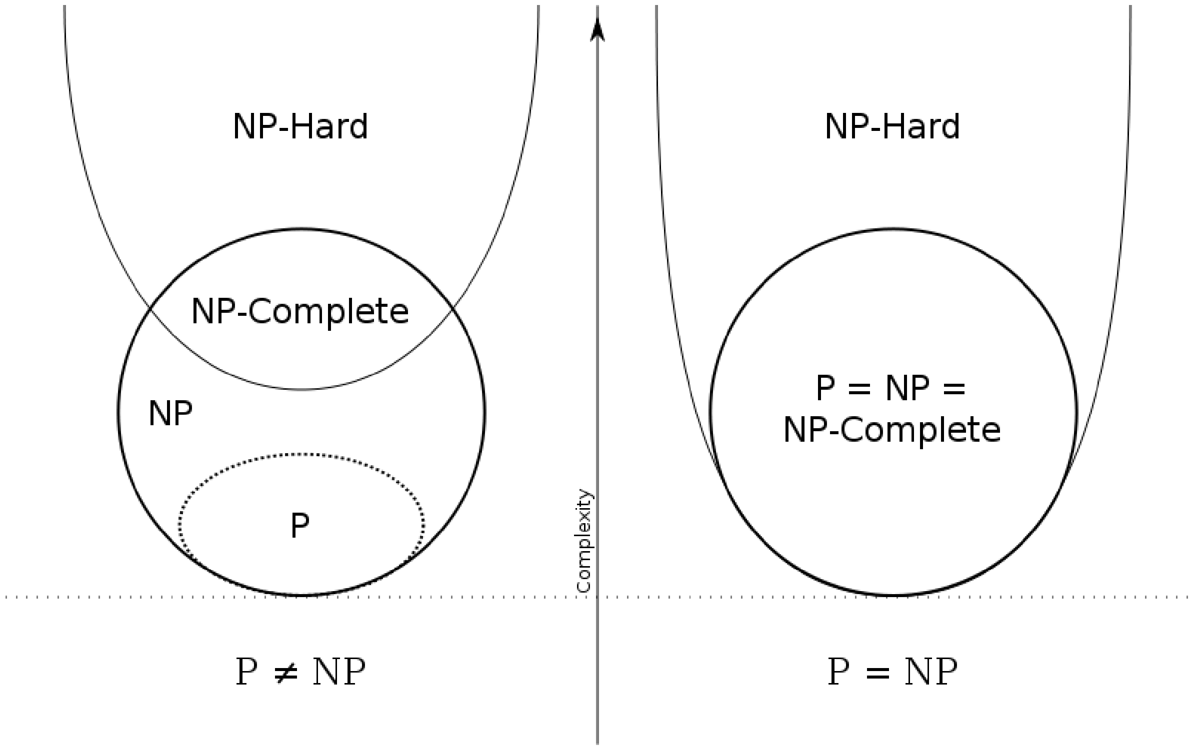


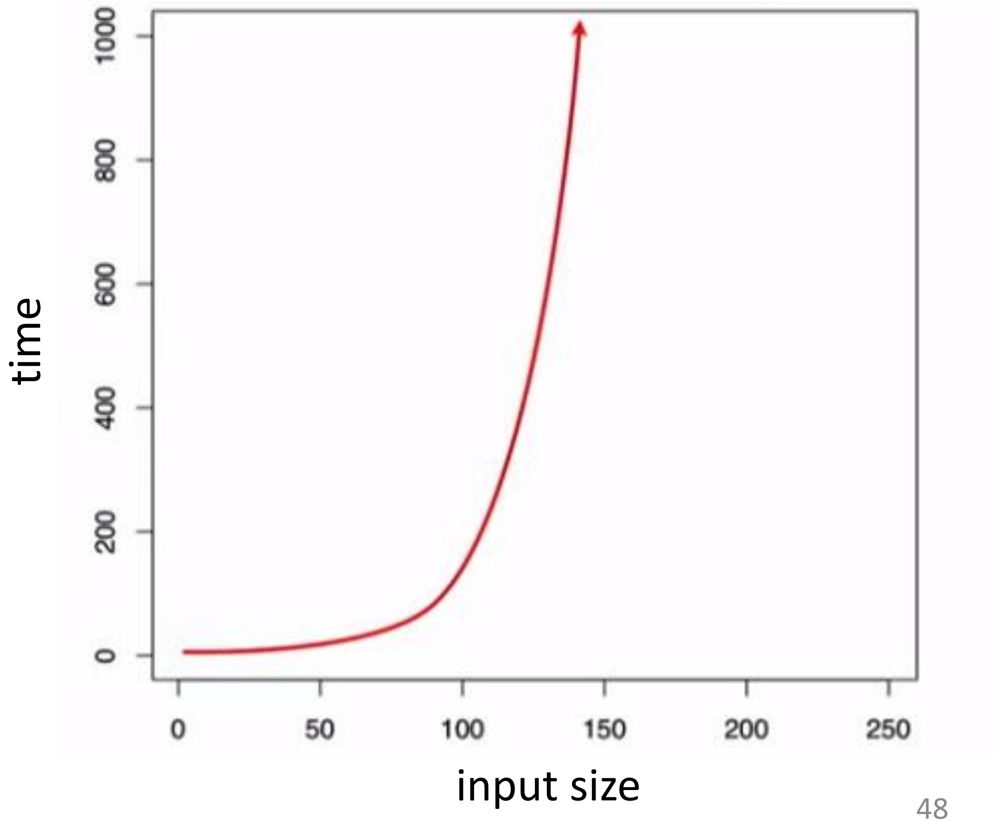


Source: [Park et al., 2019]

Np-Complete Problem

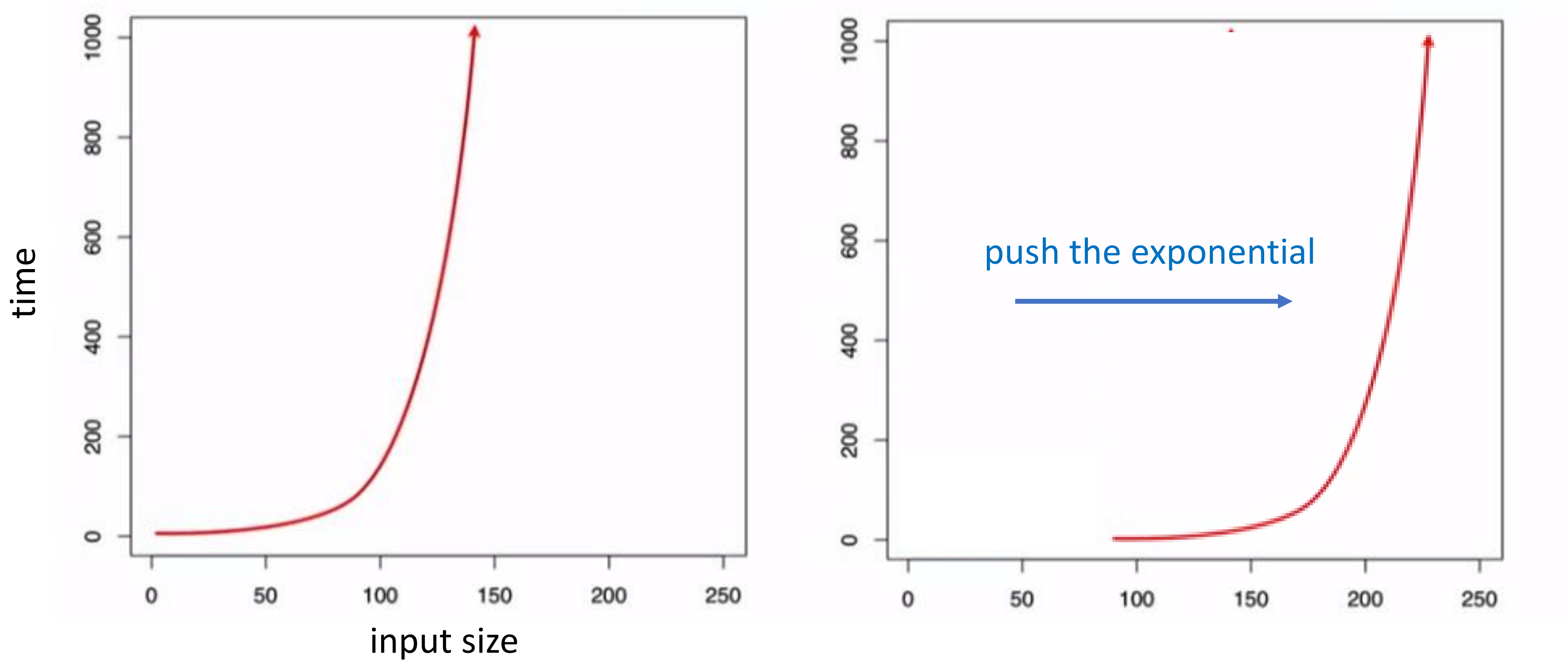
- If we have a solution we can evaluate it quickly, but finding a solution is not trivial and has exponential behaviour.





Difficult to solve

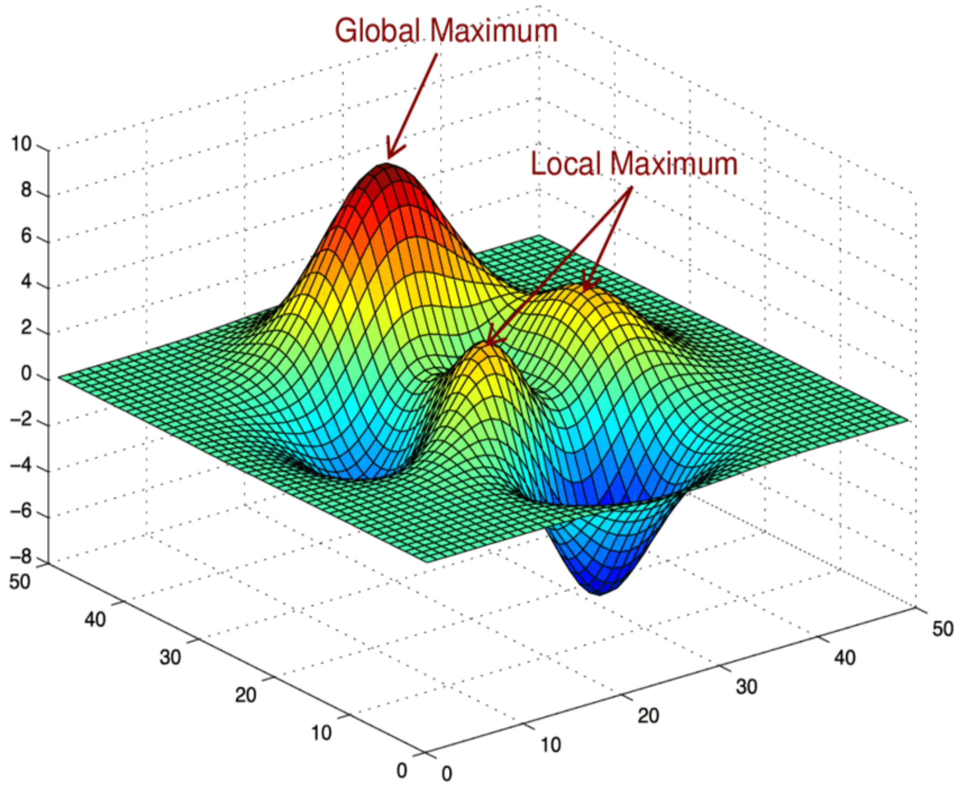
- We may be able to solve the problem for a small range of inputs, but exponential behaviour can quickly become impractical…



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



- We still have to solve the problem in some way…
- … what we can do is lower our standards
- More precisely we don’t focus on finding the best most “optimal” solution.

Example: Knapsack Problem

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

$1 Million 2kg

$1 Million 2kg

$1 Million 2kg

$7 Million 3kg

Max Capacity = 10kg

$10 Million 5kg

$10 Million 5kg

$13 Million 8kg

- Which treasure do we select?

Knapsack Problem: Attempt 1

Max Capacity = 10kg

$13 Million
8kg

$10 Million
5kg

$10 Million
5kg

$7 Million
3kg

$1 Million
2kg

$1 Million
2kg

$1 Million
2kg

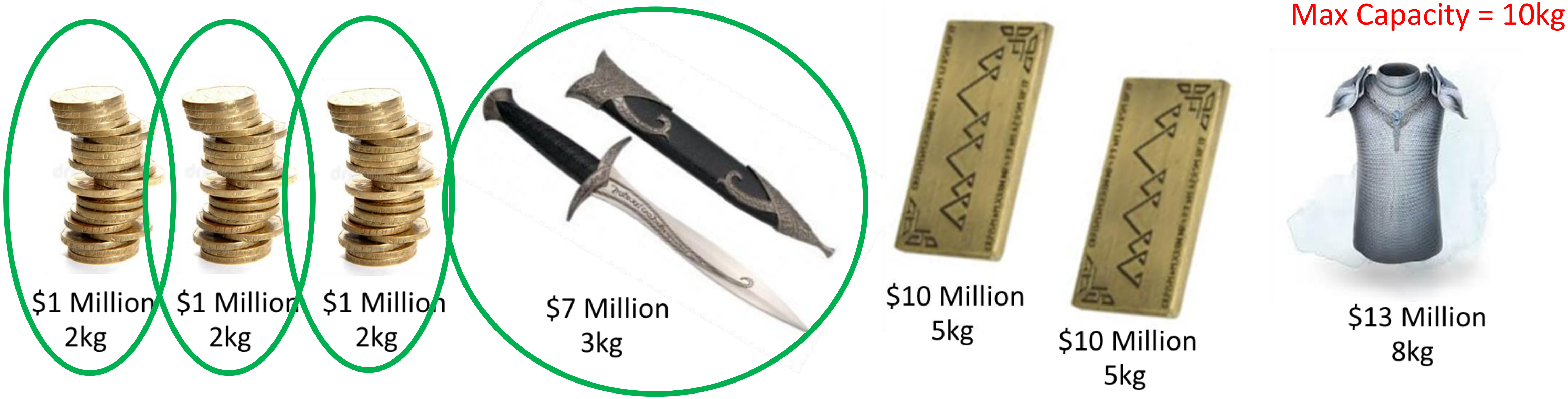
- Order the treasure by value and stuff your bag in order from most expensive to least.

$14 Million

Greedy Algorithm!

Knapsack Problem: Attempt 2

Max Capacity = 10kg

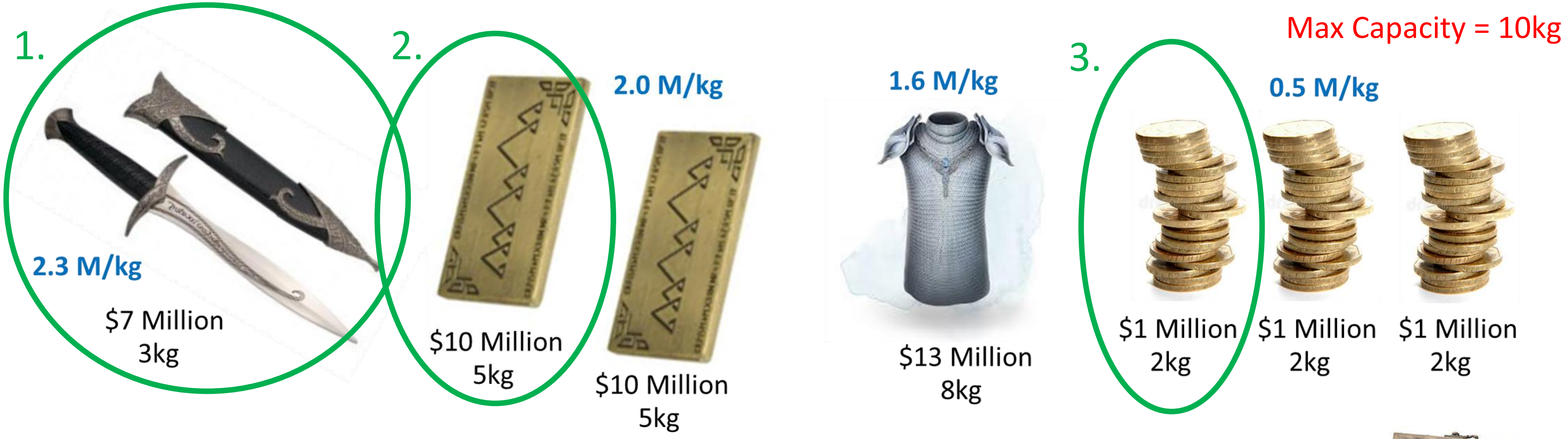


- Another intuition might be to try in the opposite order and start with the smallest weighing items hoping that you can pack more.

$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?

Max Capacity = 10kg

$1 Million
2kg

$1 Million
2kg

$1 Million
2kg

$7 Million
3kg

Max Capacity = 10kg

$10 Million
5kg

$10 Million
5kg

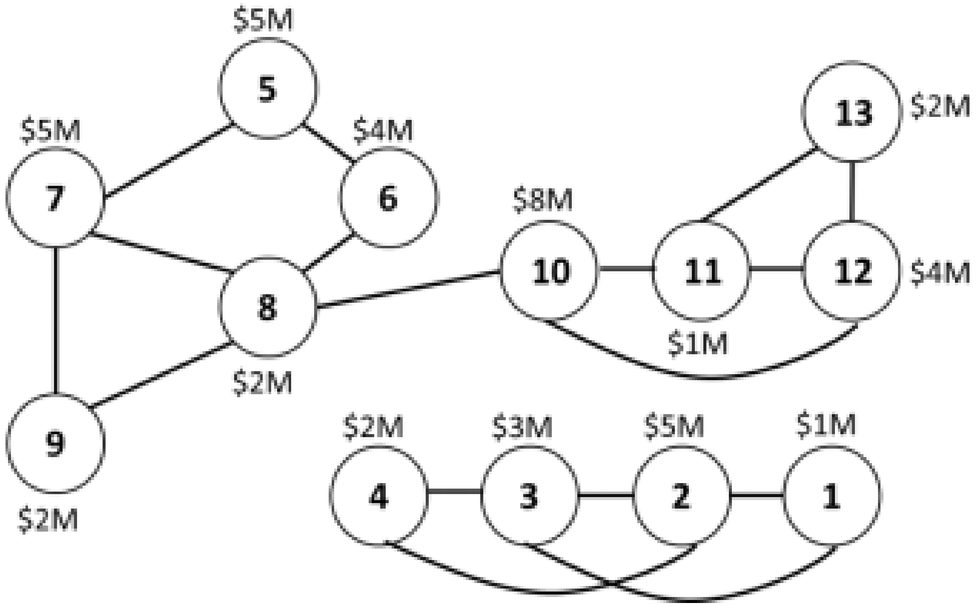
$13 Million
8kg

?

$20 Million

Set Cover Problem

- Set cover is a classical problem in combinatorics!
  - e.g. laying out Tim Horton's coffee locations at the university



The diagram represents a **system of buildings that are interconnected on the university campus.** Your goal is to select the optimal locations (1 – 13) to construct a Tim Hortons so that students can obtain a beverage and/or snack without having to traverse more than one connection.

For example, a Tim Hortons at location 1 can be accessed by students in buildings 1, 2 and 3.

Set Cover Problem

- Given a universe ***U*** of ***n*** elements (**U = {1,2,…*n*}**), a collection of subsets ***S*** ={S_1,…,S_k} of ***U***, what is the smallest sub-collection of ***S*** of which the 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

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: $\{S_1, S_2\}$, $\{S_3\}$, $\{S_1, S_3\}$, $\{S_2, S_3\}$, $\{S_1, S_2, S_3\}$.

Q: What is the cheapest combination? Why?

More Formally

Problem 5.1 SET COVER

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

Task. Solve the problem

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

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

All elements are covered (at least once)

$$\text{subject to} \quad \sum_{S : e \in S} x_S \geq 1 \quad e \in U,$$

$$x_S \in \{0, 1\} \quad S \in \mathcal{S}.$$

Variable indicating whether it's chosen or not

Source: [Alexander Souza]

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 $\mathcal{S} = \{S_1, \ldots, S_k\}$, $S_i \subseteq U$, 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***

(a) Find the most cost-effective set in the current iteration, say S.

(b) Set $x_S = 1$ and for each $e \in S - C$ set $\boxed{\text{price}(e) = c(S)/|S - C|.}$

(c) $C = C \cup S$.

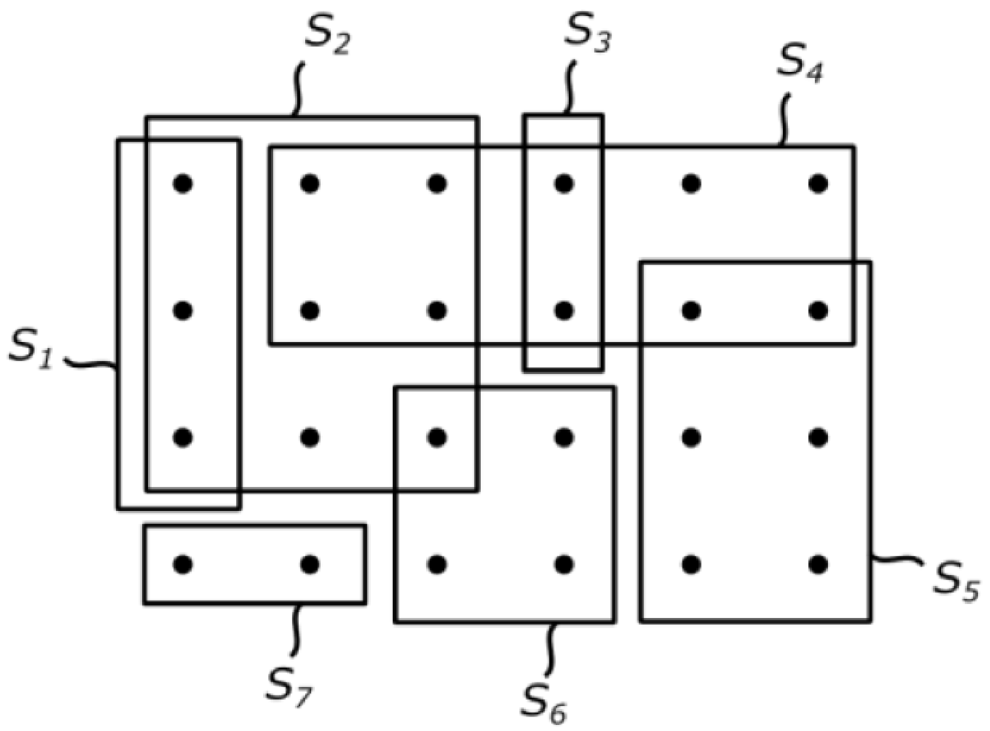
Cost of set / Elements not yet added

Step 3. Return x.

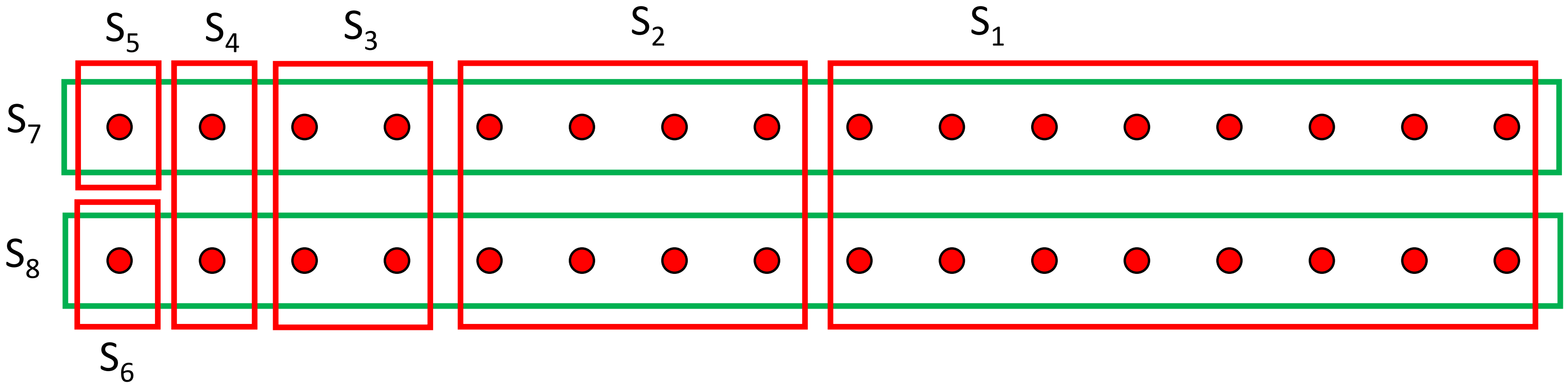
Cost-effectiveness of a set S – the average cost of covering new elements

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?



Tie Breaking



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

Source: Dave Mount

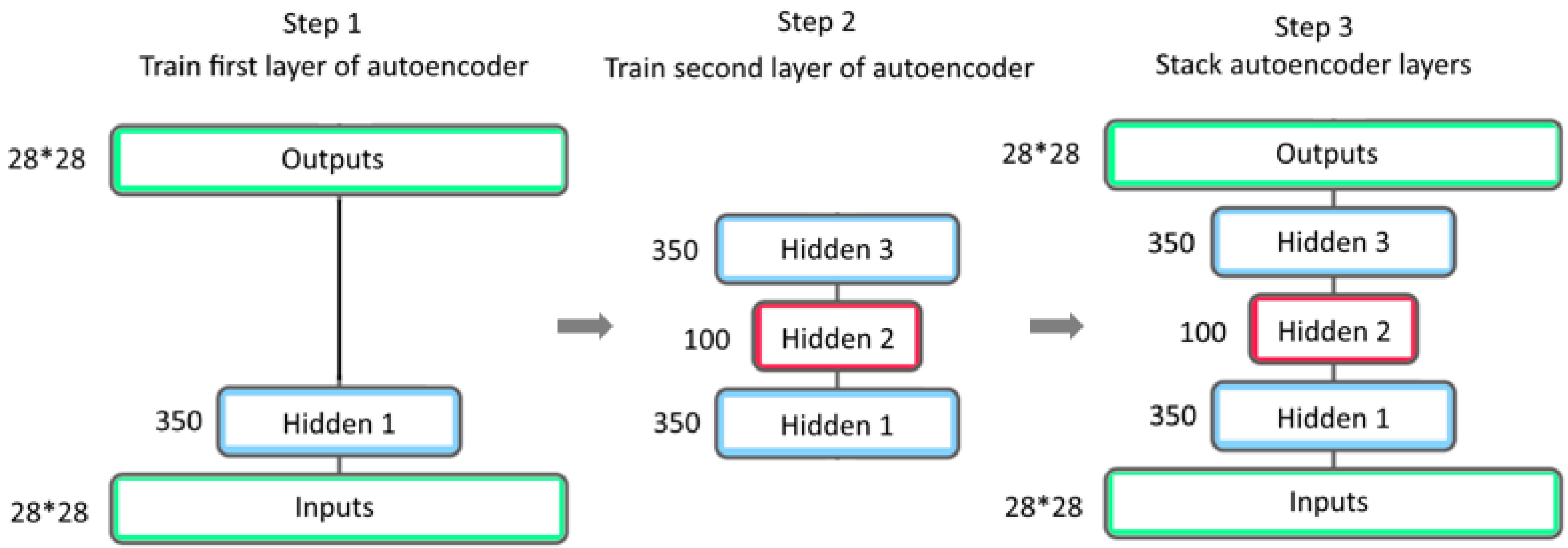
Key Takeaways

- **Optimization isn't only continuous** (gradient descent), but **can also be discrete** -> requires a different way of thinking

- Greedy algorithms are a good first approximation and generally are not that bad!

Link to Machine Learning

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



Next Time

- Please consider completing the course evaluations (available until Apr 3)
- Week 11 Q/A Support
  - Project 4 Questions
- Week 12 Lecture - Review
  - Sampling Methods
  - Hypothesis Testing
  - Final Assessment Details
  - Past Final Exam Questions
- Week 13 Lecture – Final Exam