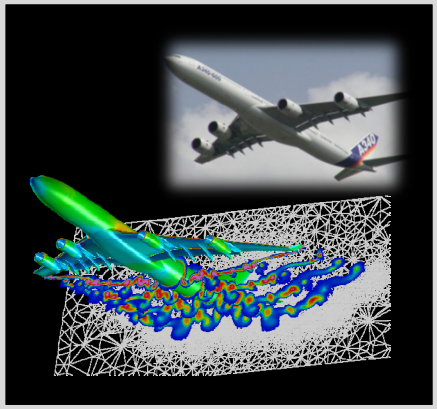


High Performance Computing

ADVANCED SCIENTIFIC COMPUTING

Dr. – Ing. Gabriele Cavallaro
Postdoctoral Researcher
High Productivity Data Processing Research Group
Juelich Supercomputing Centre, Forschungszentrum Juelich, Germany

Invited Lecture

Deep Learning for Remote Sensing Image Classification

October 07, 2019
Room V02-156

UNIVERSITY OF ICELAND
SCHOOL OF ENGINEERING AND NATURAL SCIENCES
FACULTY OF INDUSTRIAL ENGINEERING, MECHANICAL ENGINEERING AND COMPUTER SCIENCE

JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE

DEEP Projects

HELMHOLTZ RESEARCH FOR GRAND CHALLENGES

HAICU HELMHOLTZ ARTIFICIAL INTELLIGENCE COOPERATION UNIT

LECTURER

Gabriele Cavallaro

- Since 2013 working in the field of remote sensing and image processing
- Holds PhD in Electrical and Computer Engineering (University of Iceland - 2016)
  - BS and MS Degrees in Telecommunications Engineering (University of Trento – 2011,2013)
- Postdoctoral researcher at Forschungszentrum Juelich - Juelich Supercomputing Centre (since 2016)
  - Member of the High Productivity Data Processing group
  - The group focuses on application-driven parallel and scalable machine learning methods that exploit innovative high performance and distributed computing technologies
- Research interests
  - Processing and analysis of remote sensing big data
  - Parallel and scalable machine (deep) learning techniques
- Contact
  - Email: g.cavallaro@fz-juelich.de

RECOMMENDED SOURCES

- Most of the material of this lecture includes modified contents that are taken from the following sources:
  - CS231n Convolutional Neural Networks for Visual Recognition, Stanford (http://cs231n.github.io/)

  - S191: Introduction to Deep Learning, MIT (http://introtodeeplearning.com/)

OUTLINE

- Remote Sensing Backgrounds
- Pattern Recognition Systems
- Features Extraction with Shallow Learning and Deep Learning
- Convolutional Neural Networks (CNNs)
- Training Deep Networks with Backpropagation
- Distributed Deep Learning with HPC

REMOTE SENSING

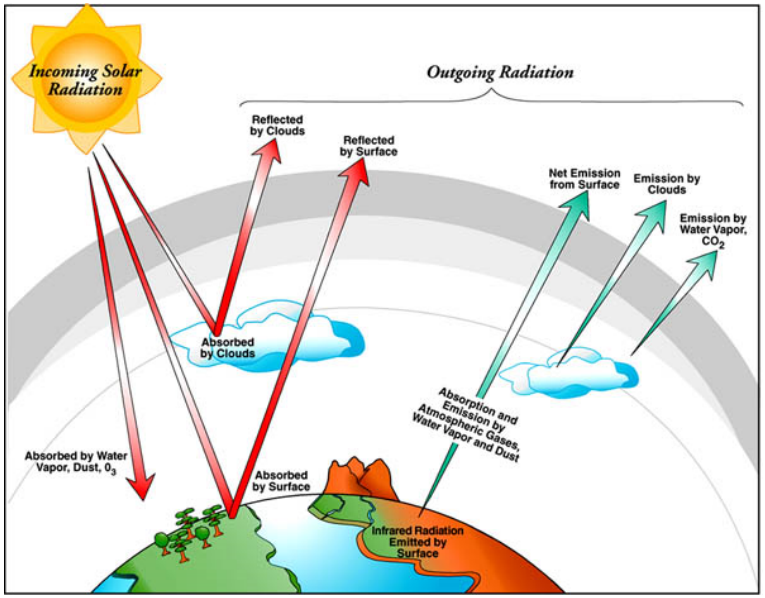
Remote (without physical contact) Sensing (measurement of information)

- Obtain information about the *atmosphere* and *surface* of Earth **without needing to be in contact with it**

- Achieved by sensing and **recording emitted or reflected energy** toward processing, analyzing, and interpreting the retrieved information for decision-making

- Measurement of **radiation of different wavelengths** reflected or emitted from distant objects or materials

- They may be categorized by class/type, substance, and spatial distribution



[2] The Earth-Atmosphere Energy Balance

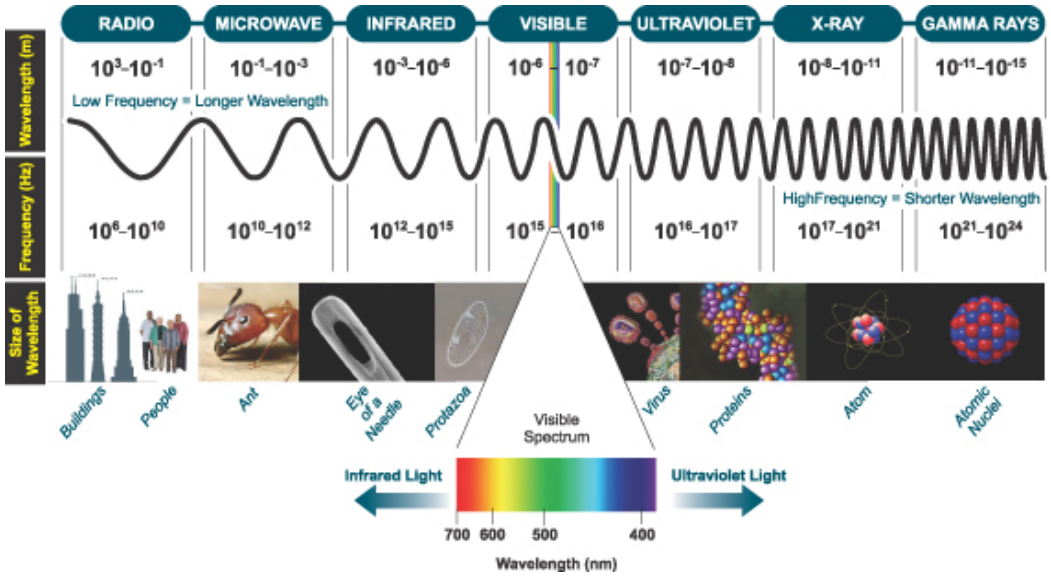
[1] Satellite (1960)

The term remote sensing was first used in the United States in the 1950s by Ms. Evelyn Pruitt of the U.S. Office of Naval Research

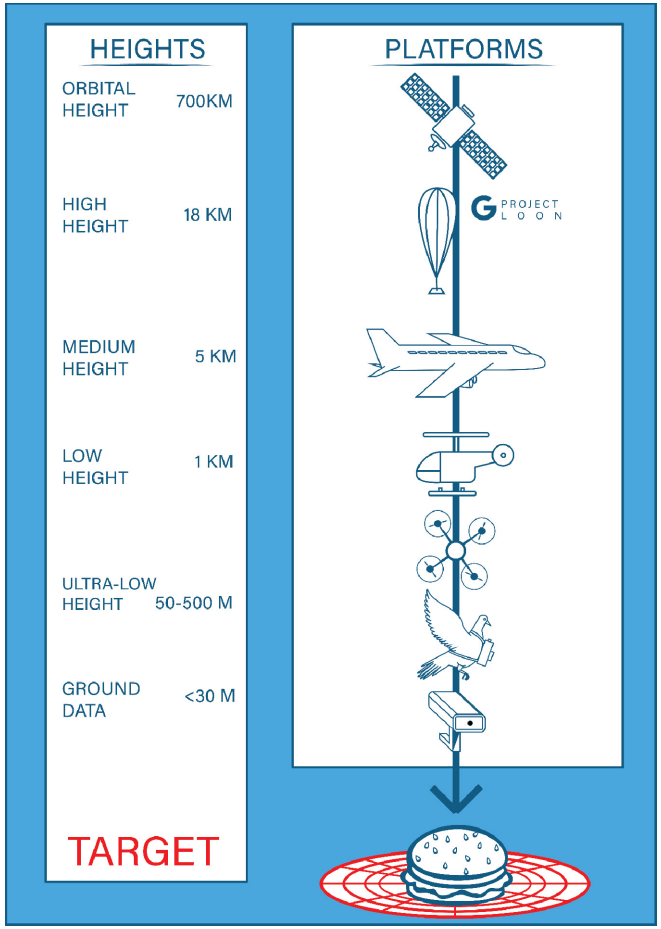
ELECTROMAGNETIC (EM) SPECTRUM

What is Sensed?

- Continuous **set of radiation** sorted according to **wavelength** (or **frequency**)
  - Subdivisions are set for convenience and by traditions within different disciplines



[3] Electromagnetic Spectrum

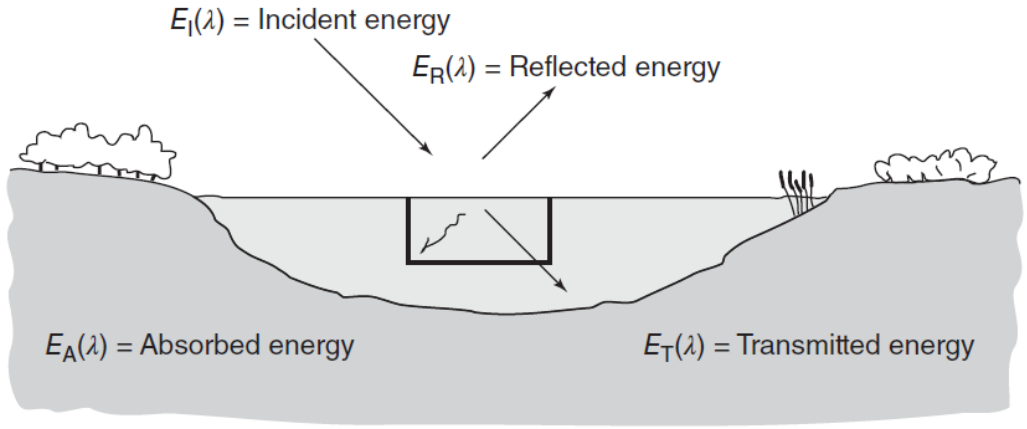


- There is neither a source nor a remote sensing system that “work” over the whole EM spectrum

ENERGY INTERACTIONS WITH SURFACE FEATURES

- Various **fractions** of the energy incident on the element **are reflected, absorbed, and/or transmitted**

$$E_I(\lambda) = E_R(\lambda) + E_A(\lambda) + E_T(\lambda)$$



$$\rho(\lambda) = \frac{E_R(\lambda)}{E_I(\lambda)} \quad \text{Reflection coefficient}$$

$$\tau(\lambda) = \frac{E_T(\lambda)}{E_I(\lambda)} \quad \text{Transmission coefficient}$$

$$\alpha(\lambda) = \frac{E_A(\lambda)}{E_I(\lambda)} \quad \text{Absorption coefficient}$$

[4] K. Tempfli et al.

- From the **conservation of energy** theorem, it can be derived

$$E_I(\lambda) = E_R(\lambda) + E_T(\lambda) + E_A(\lambda)$$

$$\rho(\lambda) + \tau(\lambda) + \alpha(\lambda) = 1$$

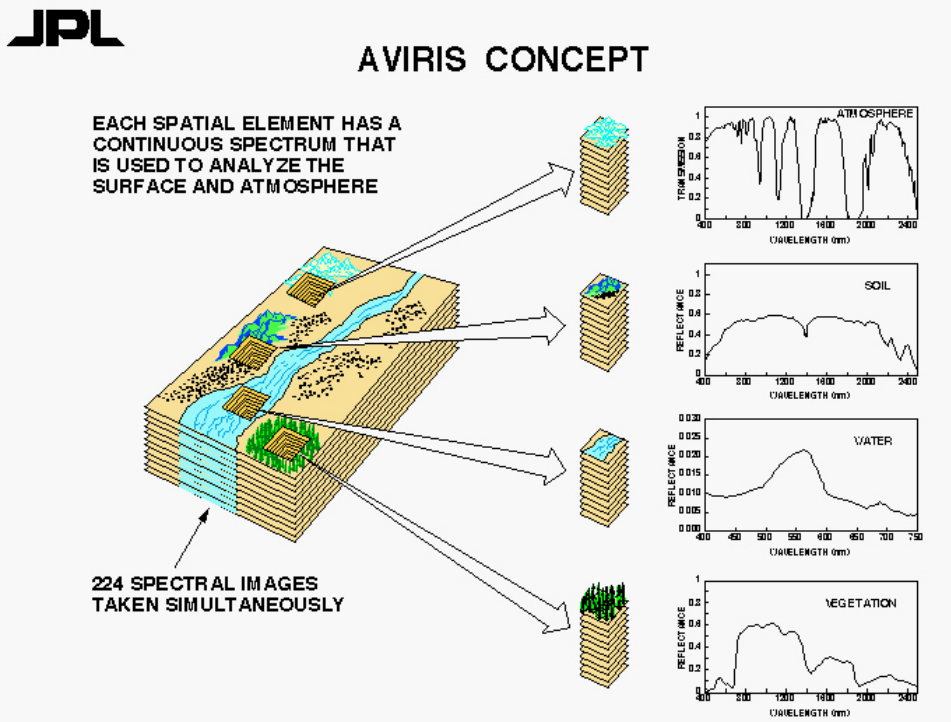
REFLECTANCE PROPERTIES OF SURFACE FEATURES

Spectral reflectance

- The reflectance properties of features can be quantified by measuring the portion of incident energy that is reflected
  - This is measured as a function of wavelength
  - It is called **spectral reflectance**

$$\rho(\lambda) = \frac{E_r(\lambda)}{E_i(\lambda)} * 100$$

- By analyzing the **spectral reflectance** it is possible to discriminate between **different land covers**

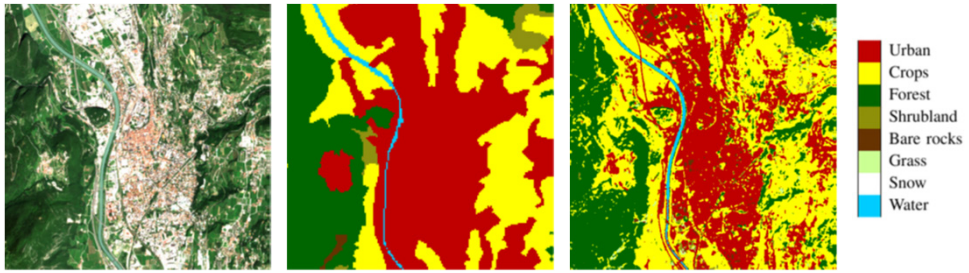


[5] AVIRIS Concept

RECOGNITION PHASE

Extract information from the image and made available to the end user

- The result depends on the specification application considered, e.g.,
  - **Thematic maps** of the territory
  - **Change maps** between 2 different dates
  - Environmental risk maps
  - 3D Topographic maps

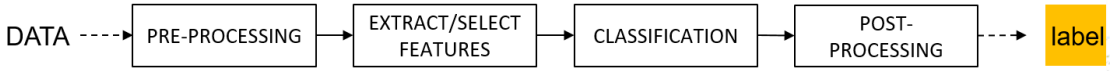


Sentinel-2 | Old map | Updated map

[6] C. Paris et al.

- Two main strategies:
  - **Not automatic** (widespread in the past and still used today)
    - With the help of photo interpreters (**photointerpretation**)
  - **Automatic recognition techniques**
    - E.g., **Pattern recognition systems**

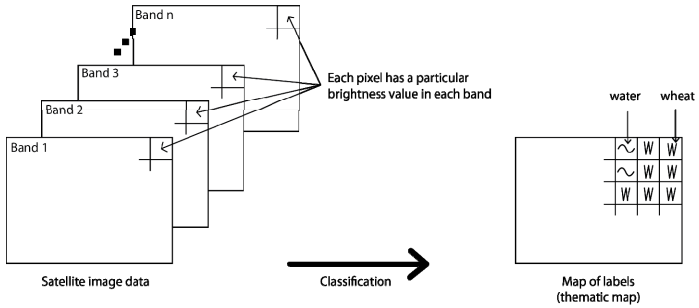
[7] Satellite Images



CLASSIFICATION TASKS

Output variable takes a class label

- **Pixel-wise** classification



- **Patch-based** classification (with single or multiple land-cover class labels)

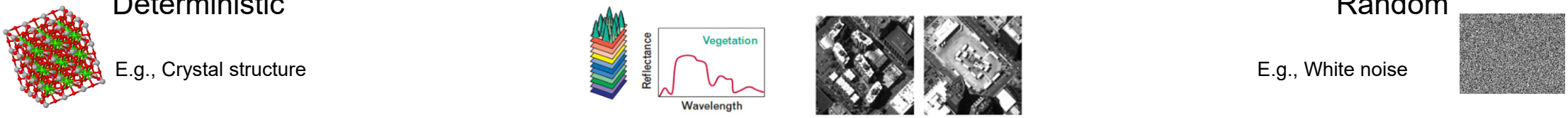
[8] P. Helber et al.

[9] G. Sumbul et al.

PATTERN RECOGNITION

Classification is a task that falls into the general category of pattern recognition

- ***Pattern***: a form, template, **composite of features**, or model (or, more abstractly, a set of rules)
  - Which can be used to make or to generate objects or parts of an object



[10] Remote Sensing

- **Pattern recognition:** automatic discovery of patterns in data through the use of tools from
  - Statistics, probability, computational geometry, **machine (deep) learning**, signal processing, and algorithm design
- Use of these patterns to take actions such as classifying the data into different **classes**

PATTERN RECOGNITION SYSTEM

Block Scheme

- **Pre-processing**
  - Atmospheric and geometrics corrections, filtering, etc.,
- **Feature extraction/selection**
  - Extraction of information parameters
  - Selection of information parameters
- **Classification**
  - Based on the information parameters previously extracted and selected
- **Post processing**

[11] Sentinel-2 satellite imagery

ACQUISTION ***DECISION***

[12] CORINE Land Cover

CLASSIFICATION PIPELINE

Two sub-tasks

1. Identification of the **features** that are unique for each class of images

2. Detection of the presence of these features

E.g, Compute vision problem

(key features in each category)

Nose,
Eyes,
Mouth

Wheels,
License Plate,
Headlights

Door,
Windows,
Steps

- Once it is established what are the features that are unique for each class
  - For a new image, if the features of a class are present, then the class can be predicted with high probability

ELEMENTS OF IMAGE INTERPRETATION

The visual interpretation of aerial and space images is a complex process

- **Representation** of features from an overhead, often unfamiliar, **vertical perspective**
- **Wavelengths outside of the visible portion** of the spectrum
- **Unfamiliar scales and resolutions**
- **High temporal resolutions:** shift from single image analysis to **time-series processing**

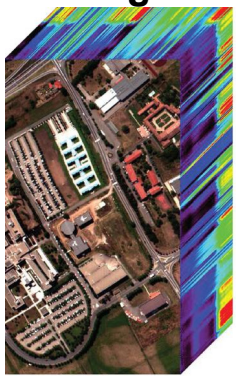
Computer Vision

Nose,
Eyes,
Mouth

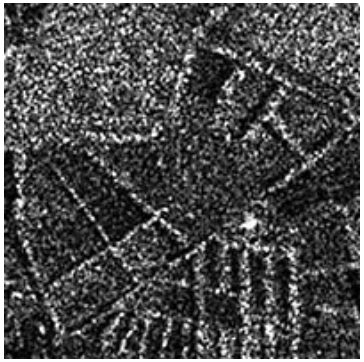
Wheels,
License Plate,
Headlights

Door,
Windows,
Steps

Remote Sensing



Hyperspectral images: hundreds of channels



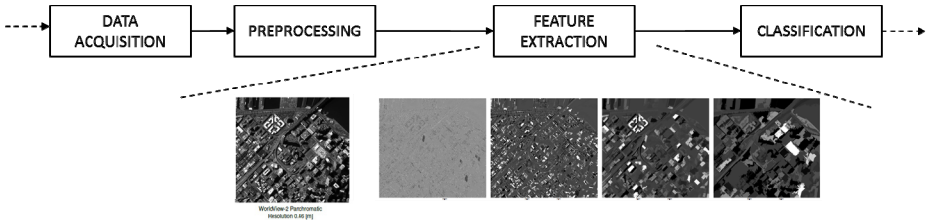
SAR images: noisy data

[13] F. Ömrüuzun et al.

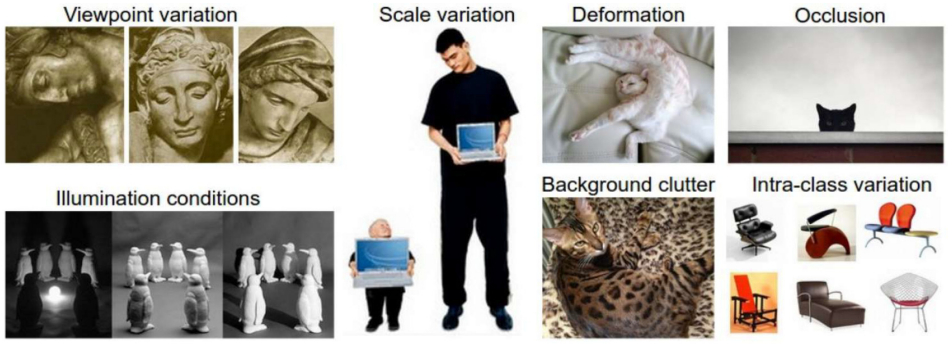
FEATURE EXTRACTION

- Both manual and automatic approaches require
  - (1) domain knowledge, (2) definition of the features and (3) detection of the features to classify

- The features are designed for a specific task
  - Based on spatial, spectral, textural, morphological content, etc.
  - E.g. Enhancing the spatial information (with attribute profiles)



- Problems
  - Images include brightness values that are under large variation

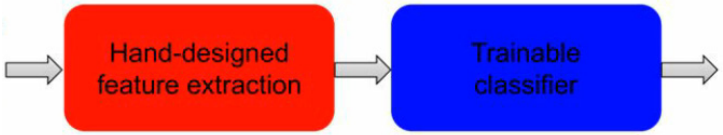


Remote sensing images exist in a variety of conditions (e.g., different seasons)

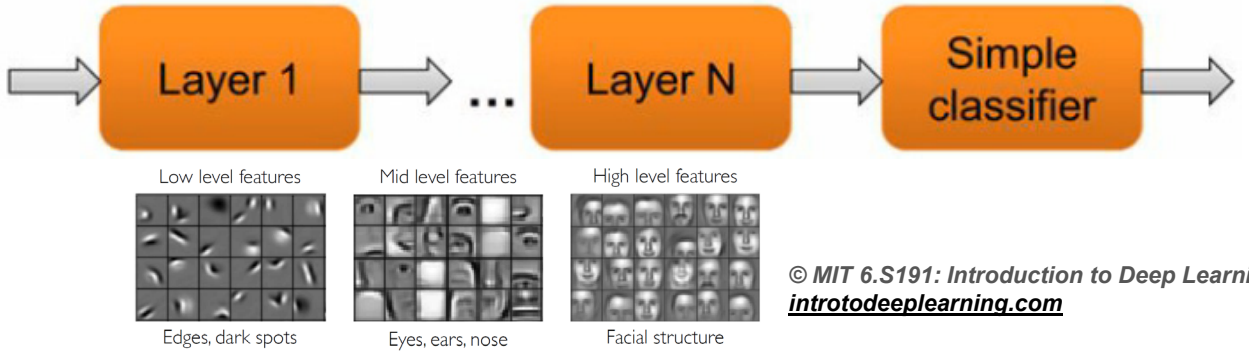
DEEP LEARNING AND SHALLOW LEARNING

Can we learn a hierarchy of features directly from the data instead of hand engineering?

- **Shallow learning**: learning networks that usually have at most one to two layers
  - E.G. Support Vector Machines (SVMs)
- They compute linear or nonlinear functions of the data (**often hand-designed features**)

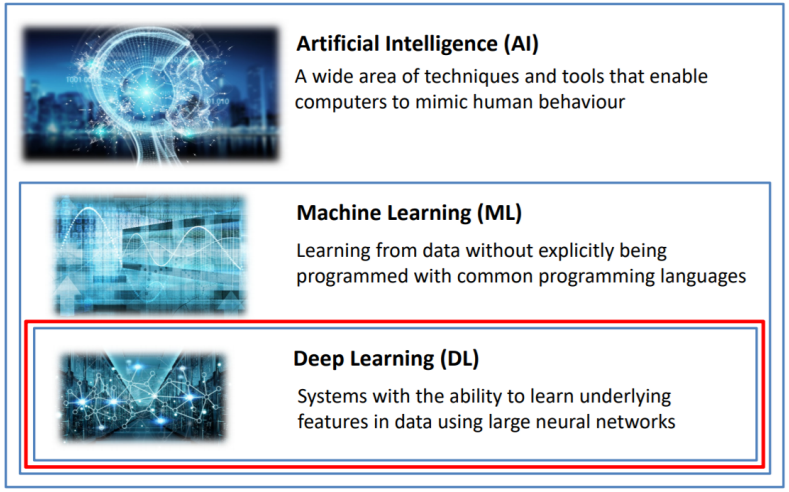


- **Deep learning:** means a deeper network with many layers of non-linear transformations
  - No universally accepted definition of how many layers constitute a “deep” learner



MACHINE (DEEP) LEARNING

- **Machine Learning:** a subfield of **Artificial Intelligence**
  - Machine learning is dealing with algorithms that can improve their functions by being exposed to data
  - Studies algorithms that improve their function from data
  - Great number of other subfields and methods
    - Kernel Machines (Gaussian Processes, Support Vector Machines, …)
    - Decision Trees (Random Forests, …)
    - and much more ...



Prerequisites for Machine Learning

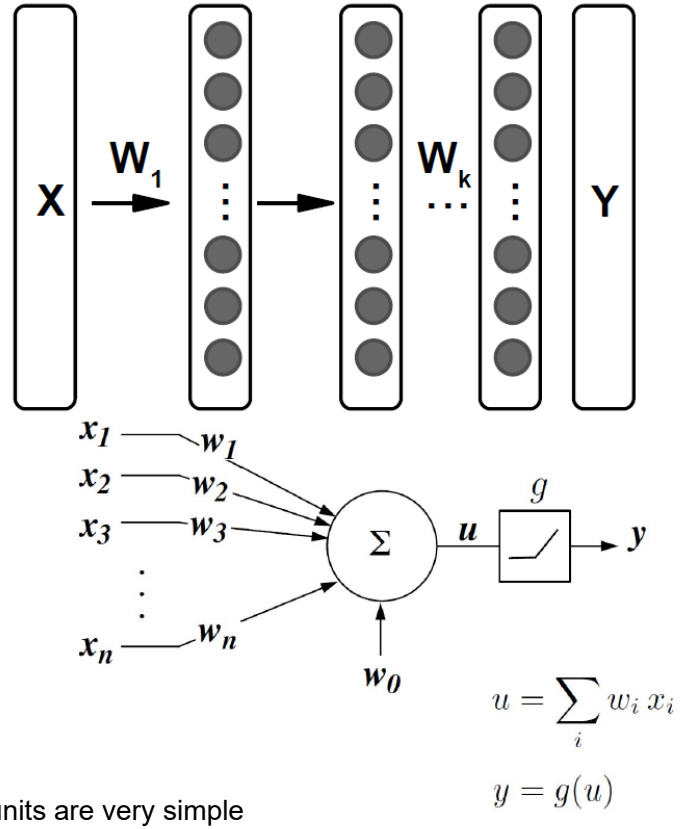
1. Some pattern exits
2. No exact mathematical formula
3. **Data exists**
   - Idea: **learning from data**
   - Challenges:
     - Data is often complex
     - Learning from data require processing time

- **Deep Learning**: a subfield of **Machine Learning**

DEEP LEARNING

- **Deep Learning**: using a generic, flexible model family
  - **"Neural" Networks** with multiple layers
  - Based on stacked, repetitive operations
  - Executed by simple, generic units
  - With parameters ("weights") adapted from incoming data

- **Deep Neural Networks**
  - Most models can be cooked down to stacked repetitive operations
  - They are executed by simple units
  - **Linear summation + non-linear transfer function**

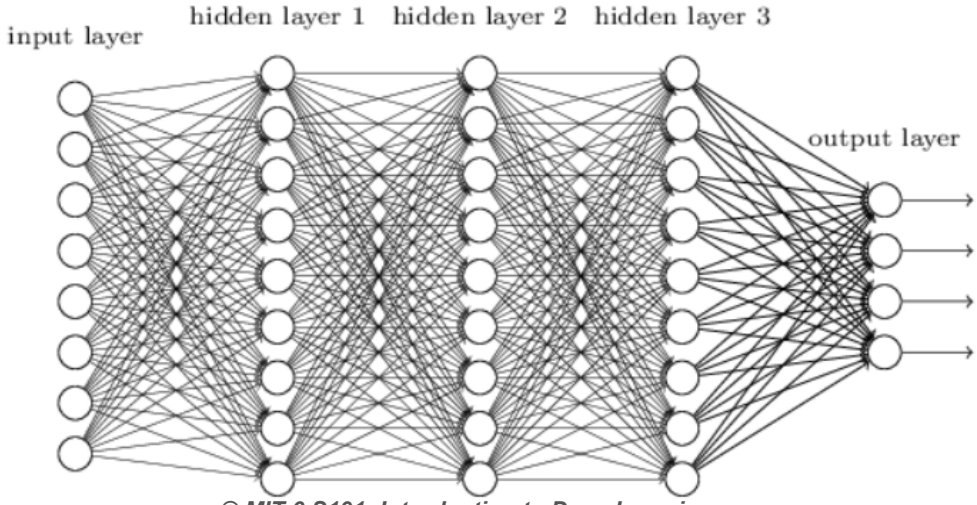


- The units are very simple
- The complexity arise when combining more of them
- The are many weights (connections) that are adjustable from the data (they are not fixed)

FULLY CONNECTED NEURAL NETWORK (FCNN)

Architecture Overview

- Receives an input a **single vector** and transforms it through a series of **hidden layers**
- Each hidden layer is made up of a set of **neurons** that have **learnable weights** and **biases**
  - Each neuron receives some inputs, performs a dot product and **follows it with a non-linearity**
  - Each neuron is **fully connected** to all neurons in the previous layer
  - Neurons in a single layer function completely independently and **do not share any connections**
- The last fully-connected layer is the “output layer” and in classification settings it represents the class scores



Bias: additional set of weights that require no input, and this it corresponds to the output of the network when it has zero input (not tied to any previous layer)

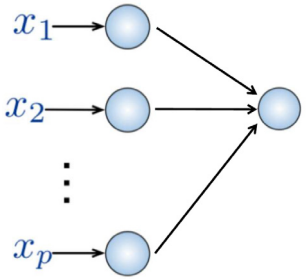
HOW TO USE SPATIAL STRUCTURE IN THE INPUT

To inform the architecture of the network?

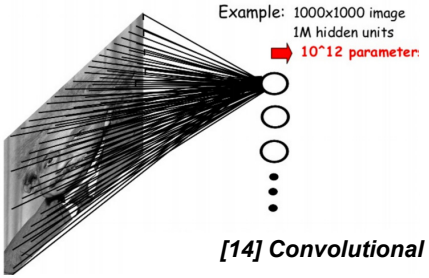
- **Fully connected:**
  - Connect neuron in hidden layer to all neurons in input layer
  - **No spatial information**

x_1	x_2	x_3
x_4	x_5	x_6
x_7	x_8	x_9

Input:
- 2D image is vectorized



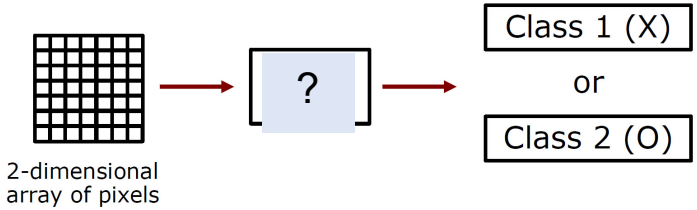
- **FCNN don't scale well to full images**
  - Moreover, when adding several neurons, the parameters grows quickly
  - The fully connectivity is wasteful, and the huge number of parameters can lead to **overfitting**



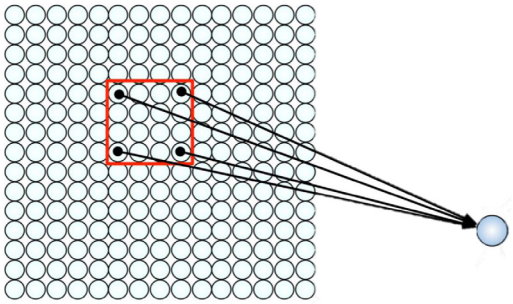
[14] Convolutional Neural Networks

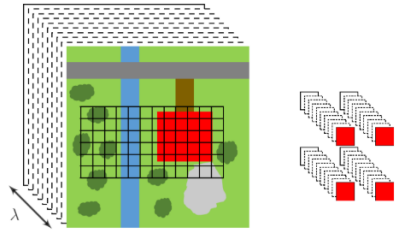
INPUT SPATIAL STRUCTURES

- Need of an architecture that makes the explicit assumption that the inputs are **images**



- **Idea**
  - Connect patches of input to neurons in hidden layer
  - Neuron connected to region of input
  - Only “sees” these values
  - Reduce the number of weights



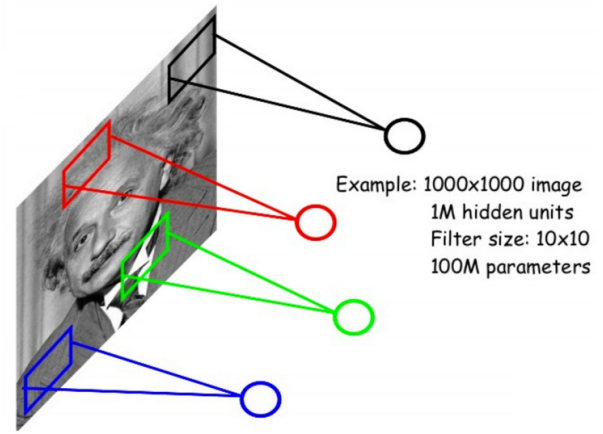


Exploit the neighboring relationship

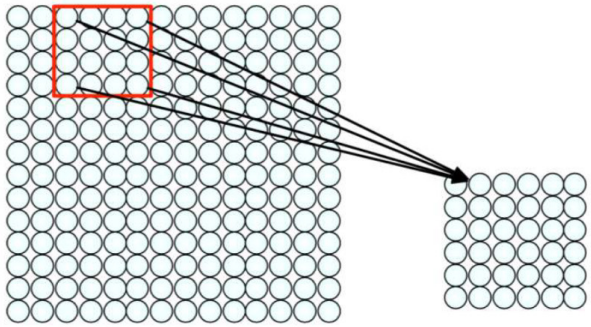
APPLY FILTERS TO EXTRACT FEATURES

With Convolution

1. Apply a set of weights – a filter – to extract **local features**
2. Use **multiple filters (kernels)** to extract different features
3. Spatially share parameters of each filter
   - Features that matter in one part of the input should matter elsewhere



[14] Convolutional Neural Networks



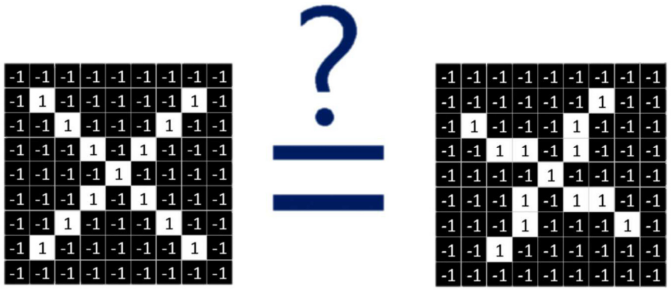
- Filter of size 4x4 : 16 different weights
- Apply this same filter to 4x4 patches in input
- Shift by 2 pixels for next patch

This "patchy" operation is **convolution**

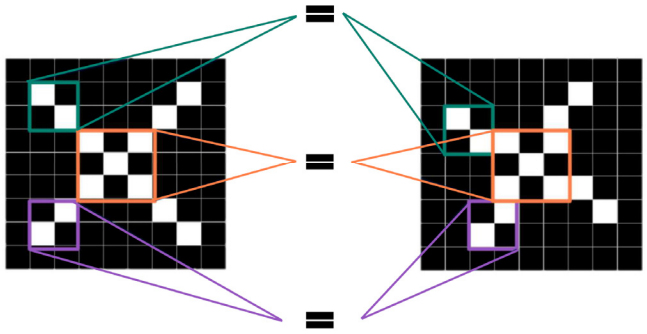
REVIEW THE CONVOLUTION

Example of classification task: classify X from a set of binary images

- For classification it would not be possible to simply compare the two matrices to check is they are equal
- The classier needs to classify an X as an X even if its shifted, shrunk, rotated, deformed, etc..

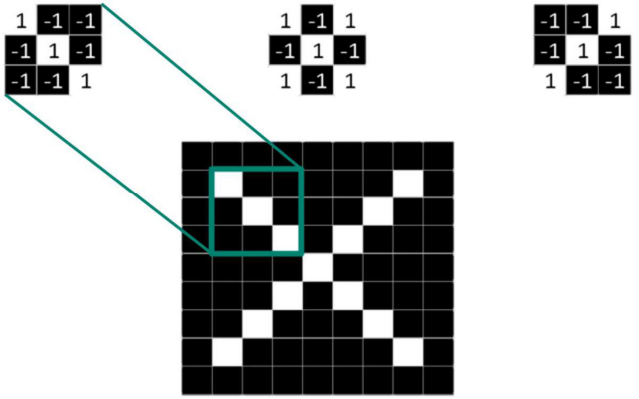


- **Effective approach**: compare the images piece by piece
- Search the important parts that define an X as an X (the meaningful **features**)



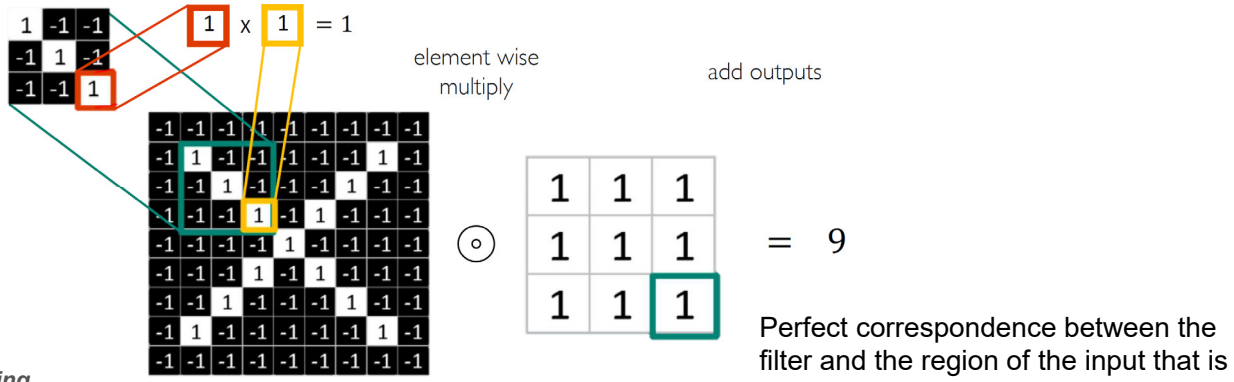
REVIEW THE CONVOLUTION

Filters to detect X features



- Think about each feature as small patches
- Use filters of weights for the convolutional operations
  - To detect the corresponding features

- Convolution preserves the spatial relationship between pixels
  - By learning image features in small squares of the input



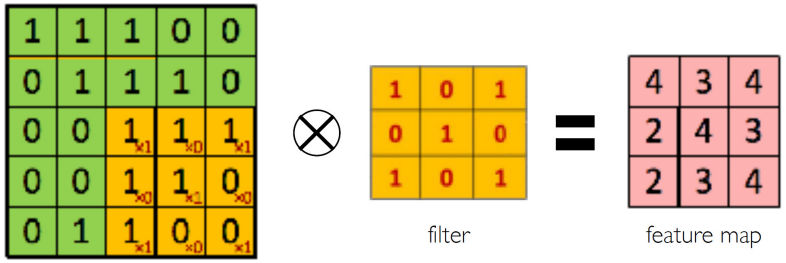
Perfect correspondence between the filter and the region of the input that is convolved

REVIEW THE CONVOLUTION

Producing feature maps

- **Feature map:** reflects where in the input was activated by the applied filter

- E.g., Slide the 3x3 filter over the input image,
  element-wise multiply, and add the outputs

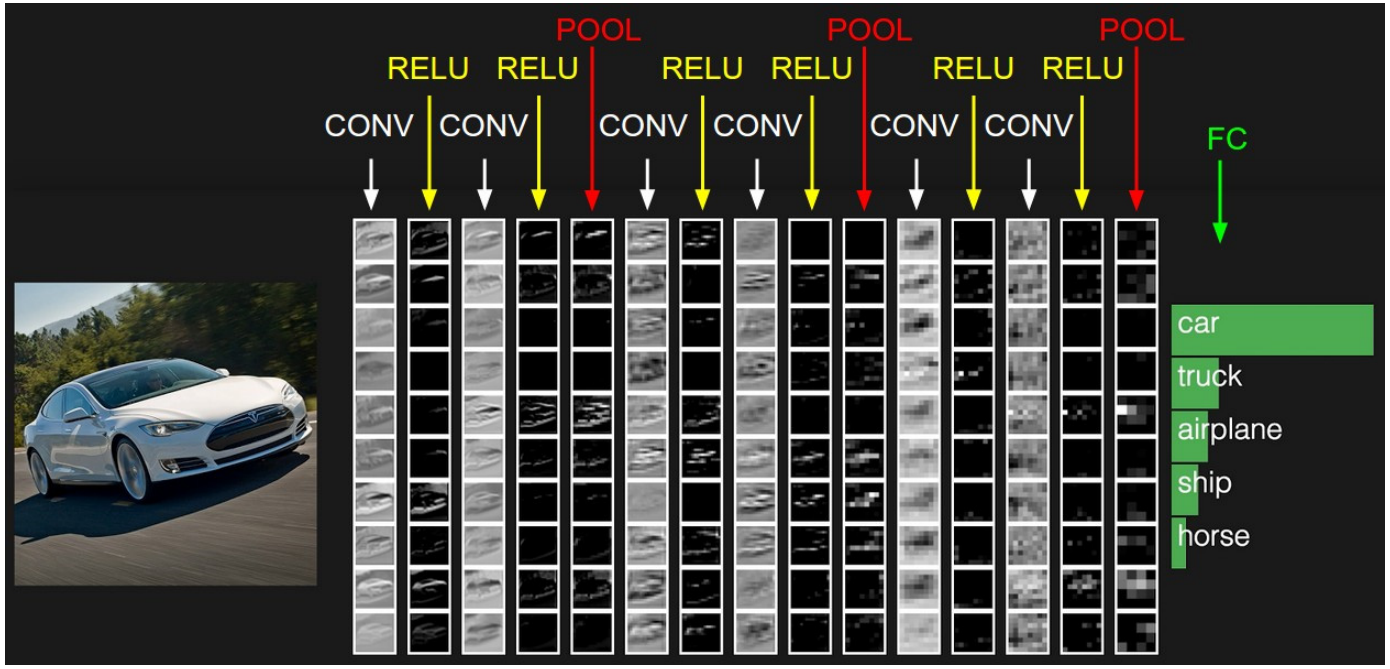


- Different filters can be used to produce different filter maps
  - E.g., With 3 different convolutional filters (i.e., different weights)

ACTIVATIONS (FEATURE MAPS)

With CNN architectures

- The initial volume stores the raw image pixels
- Volume of **activations** shown as a column (i.e., lay out each volume's slices in rows)
- The last volume holds the scores for each class

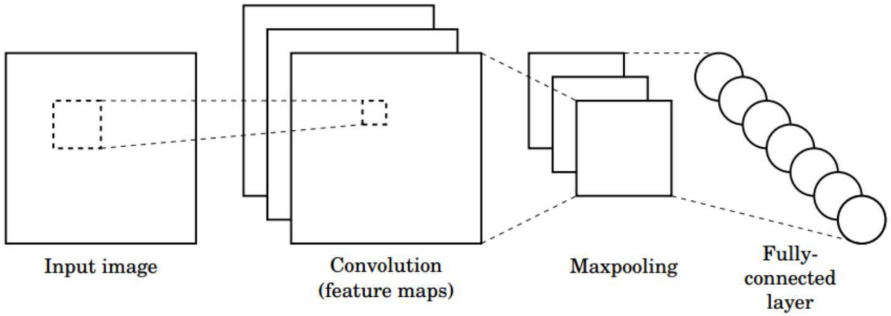


[15] CS231n

CONVOLUTIONAL NEURAL NETWORKS

Three main operations

- Sequence of layers that transforms one volume of activations to another through a differentiable function
- Three main types of layers to build the architecture:
  - **Convolutional Layer**
    - Apply filters with learned weights to generate feature maps
    - Introduce non-linearity with an **activation function** (e.g., ReLU)
  - **Pooling Layer**
    - Downsampling operation on each feature map
  - **Fully-Connected Layer**
    - Perform classification

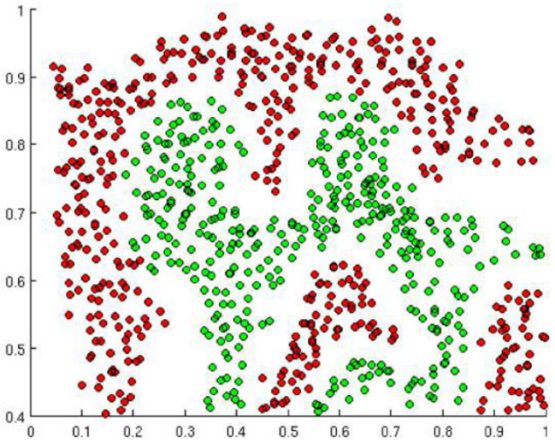


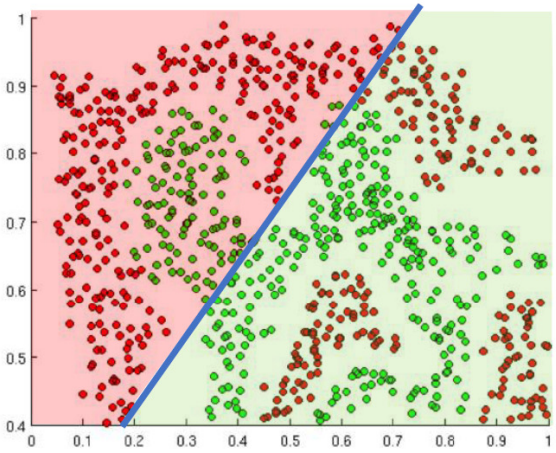
- Train the model with the **training set**
- Learn the **weights** of filters in convolutional layers (i.e., feature maps)

IMPORTANCE OF ACTIVATION FUNCTIONS

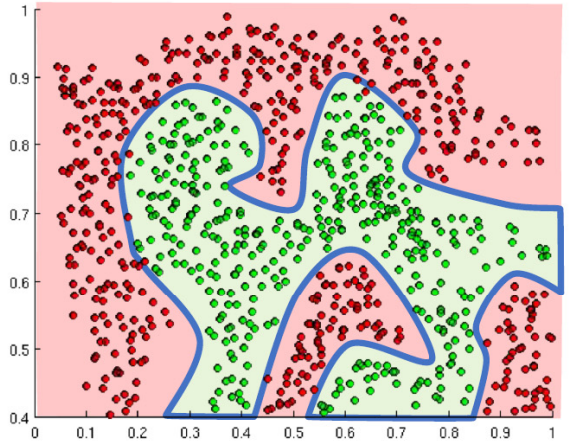
Introduce non-linearities into the network

- E.g., How to build a Neural Network to distinguish green vs red points?





Linear Activation functions produce linear decisions no matter the network size



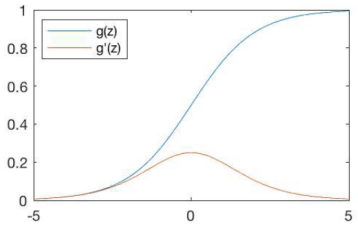
Non-linearities allow us to approximate arbitrarily complex functions

INTRODUCING NON-LINEARITY

Input images are highly non linear

- **The activation function is applied after every convolution operation**
- Each activation function (or non-linearity)
  - Takes a single number
  - And performs a certain fixed mathematical operation on it
- There are several activation functions you may encounter in practice

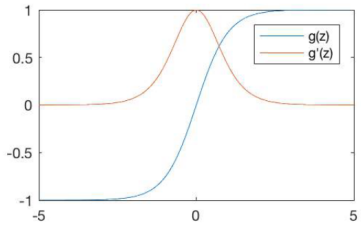
Sigmoid Function



$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

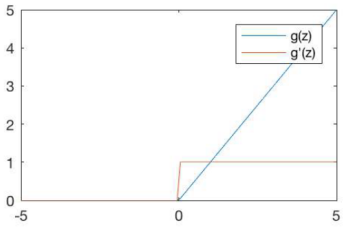
Hyperbolic Tangent



$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

Rectified Linear Unit (ReLU)



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

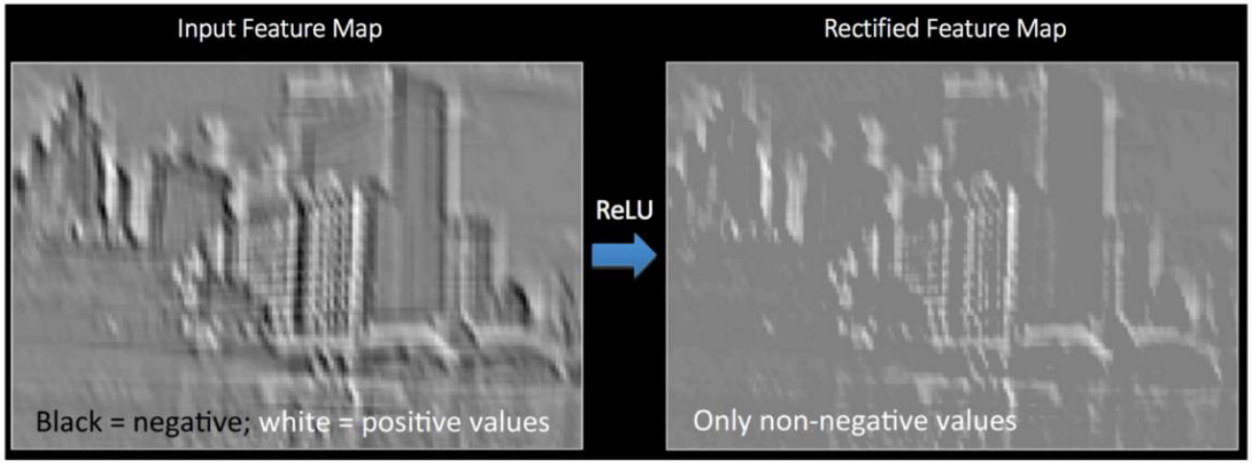
$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

[15] CS231n

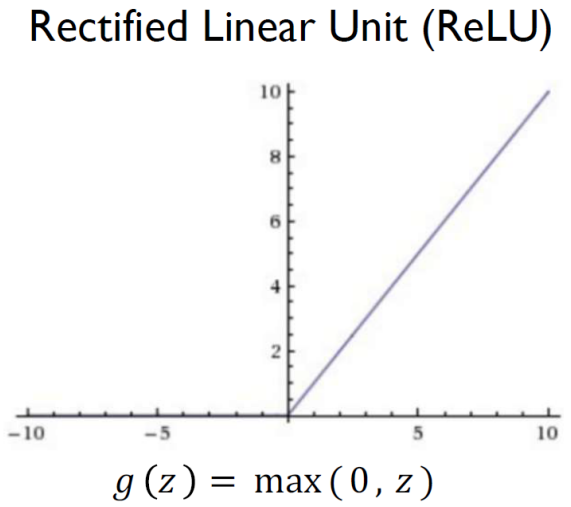
RECTIFIED LINEAR UNIT

Has become very popular in the last few years

- Compared to the sigmoid/tanh functions
  - (+) It was found to greatly accelerate the convergence of **Stochastic Gradient Descent (SGD)**
  - (+) Simply thresholding a matrix of activations at zero (no expensive operations, e.g., exponentials)



[15] CS231n

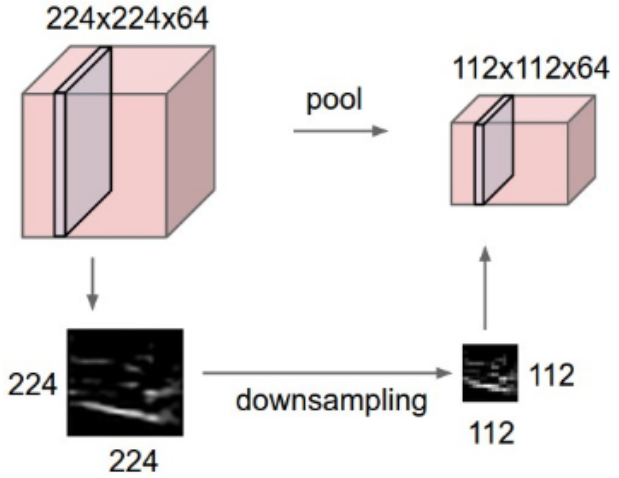


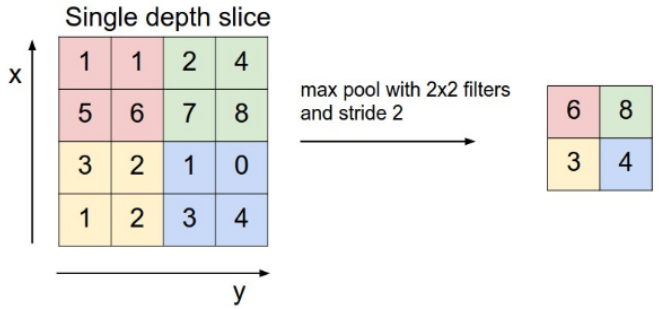
[16] A. Krizhevsky et al.

POOLING LAYER

Reduce the dimensionality and preserve spatial invariance

- It is common to periodically insert a **Pooling layer** in-between successive **Convolutional layers**
- Its function is to progressively reduce the spatial size of the representation
  - To reduce the amount of parameters and computation in the network
  - Hence to also **control overfitting**
- **Introduces zero parameter**s since it computes a fixed function of the input

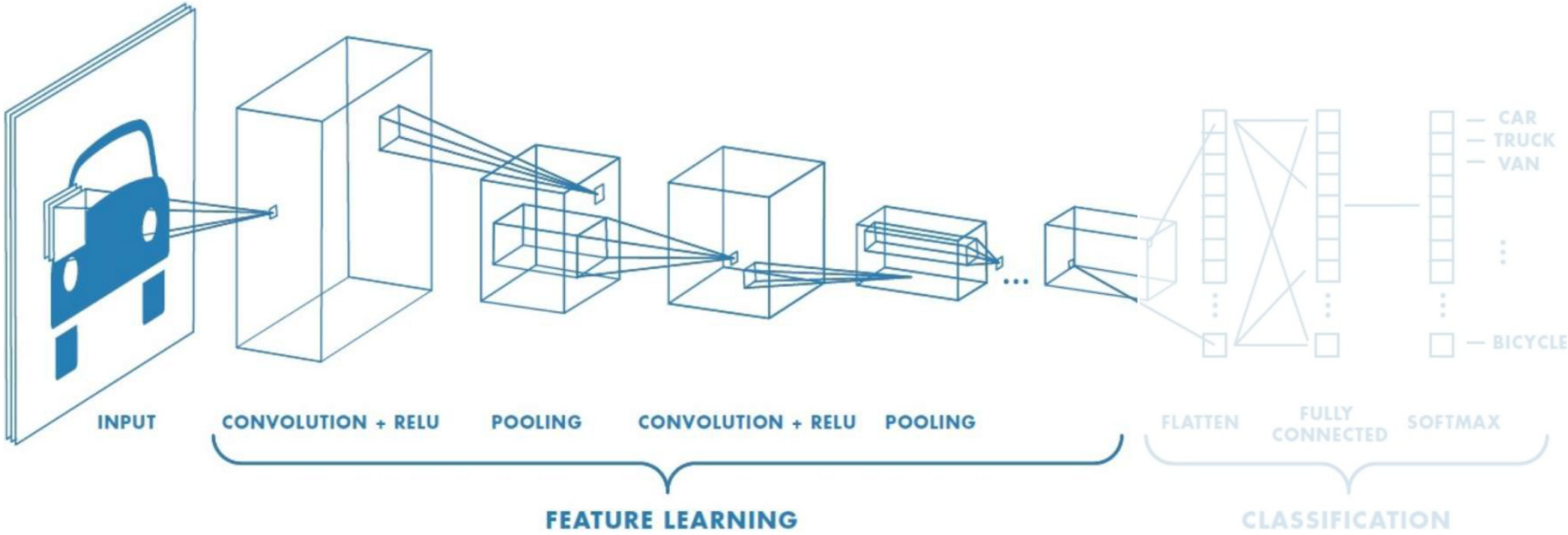




[15] CS231n

CNN FOR CLASSIFICATION

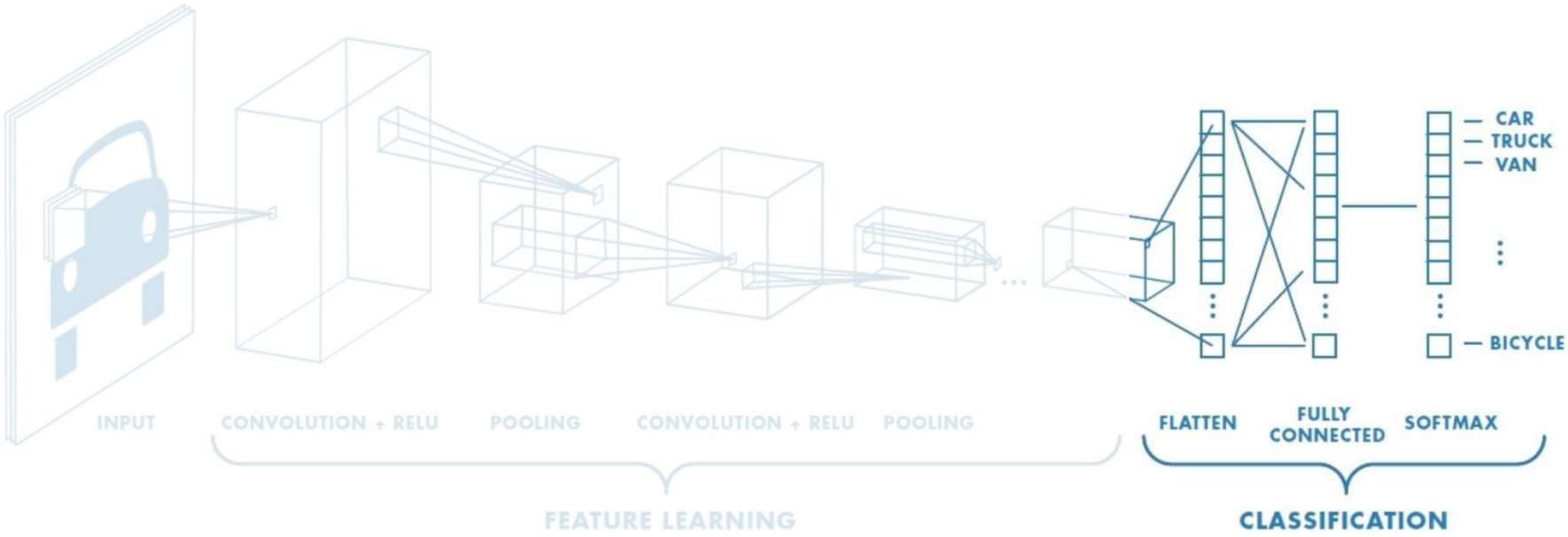
Feature learning pipeline



1. Learn features in input image through **convolution**
2. Introduce non-linearity through **activation function** (real-world data is non-linear!)
3. Reduce dimensionality and preserve spatial invariance with **pooling**

CNN FOR CLASSIFICATION

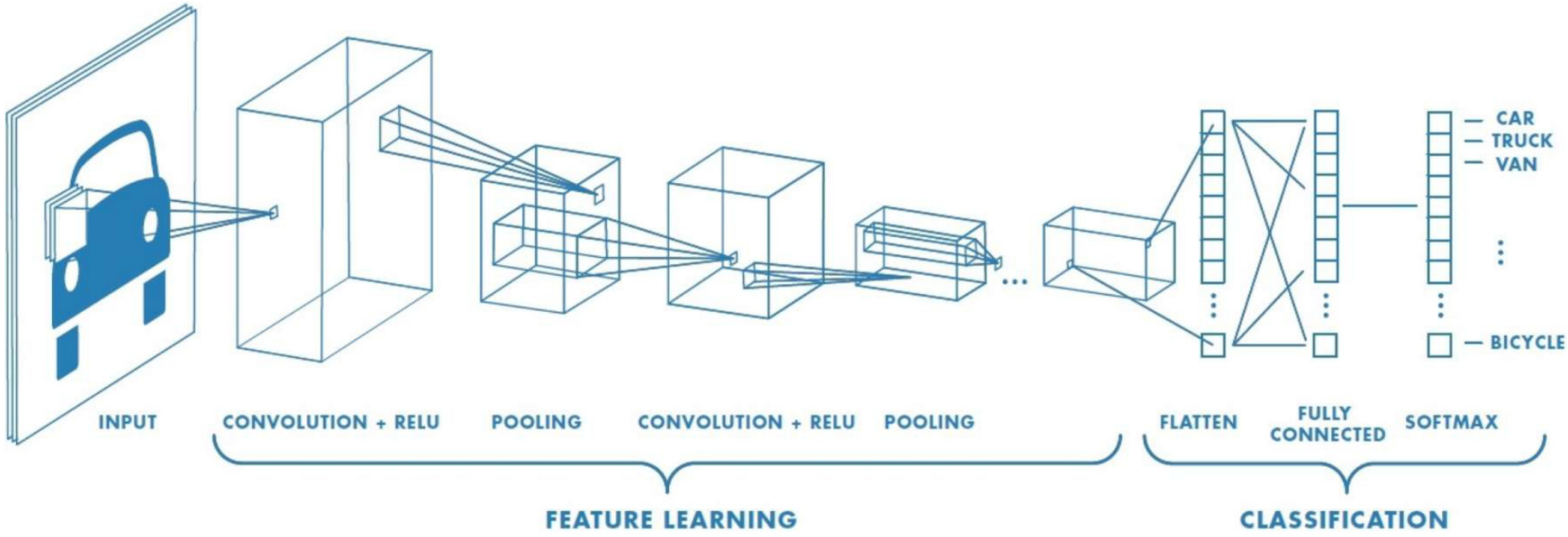
Class Probabilities



- Convolutional and Pooling layers output high-level features of input
- **Fully connected layer** uses these features for classifying input image
- Express output as **probability** of image belonging to a particular class

CNN FOR CLASSIFICATION

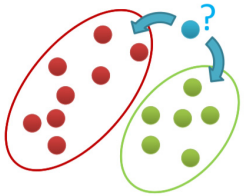
Training with Backpropagation



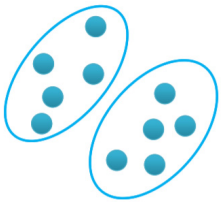
- Learn weights for convolutional filters and fully connected layers

MACHINE LEARNING: FORMS OF LEARNING

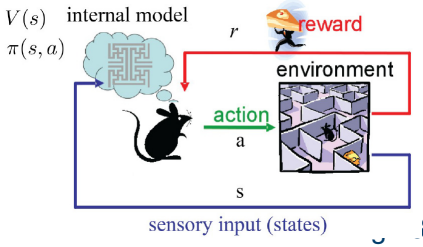
- **Supervised learning:** correct responses $\boldsymbol{Y}$ for input data $\boldsymbol{X}$ are given
  - $\boldsymbol{Y}$ "teacher" signal, correct "outcomes", "labels" for the data $\boldsymbol{X}$
  - Usually: estimate unknown $f: X \to \mathrm{Y}, \boldsymbol{Y} = f(\boldsymbol{X}; \boldsymbol{W})$
  - Classic frameworks: **classification**, **regression**



- **Unsupervised learning:** only data **X** are given
  - Find "hidden" structure, patterns in the data;
  - In general, estimate unknown probability density $\mathrm{p}(\boldsymbol{X})$
  - Broad class of latent ("hidden") variable models
  - Classical frameworks: **clustering**, **dimensionality reduction**



- **Reinforcement learning:** data $\boldsymbol{X}$, including (sparse) **reward** $\mathrm{r}(\boldsymbol{X})$
  - Discover actions $\boldsymbol{a}$ that minimize total future reward R
  - **Active** learning: experience $\boldsymbol{X}$ depends on choice of $\boldsymbol{a}$
  - Estimate $p(\boldsymbol{a}|\boldsymbol{X}), p(r|\boldsymbol{X}), V(\boldsymbol{X}), Q(\boldsymbol{X}, \boldsymbol{a})$ - future rewards predictors



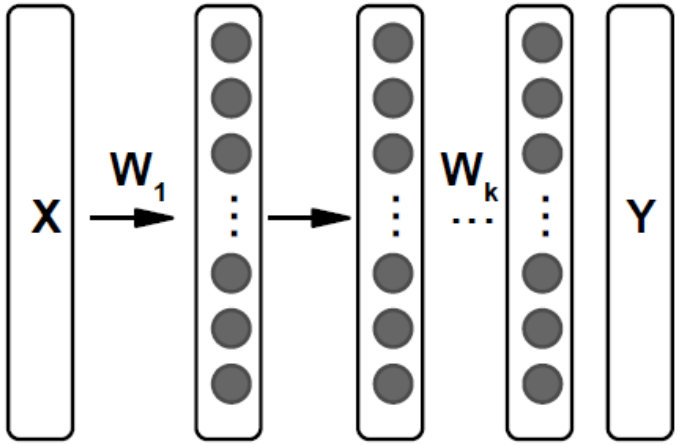
For all holds: Define a **loss** $\mathcal{L}(\mathbf{W})$, optimize by tuning parameters **W**

DEEP LEARNING

Learning as an Optimization Procedure

- Learning as **optimization problem**
  - Defining **loss** $\mathcal{L}$(cost) function sets up optimization problem
  - Optimization can be done in various ways
  - Deep Neural Networks: **gradient descent** methods dominate
  - **Automatic differentiation**

- $f(\boldsymbol{X};\boldsymbol{W}), \mathrm{P}(\mathbf{Y}|\mathbf{X},\mathbf{W})$: unknown
  - -> Learn from data!
- Define loss $\mathcal{L}(f(\boldsymbol{X};\boldsymbol{W}), Y)$
- Minimize loss: adapt $\boldsymbol{W}$ from the data
- Data-driven Optimization
  - Using gradient descent

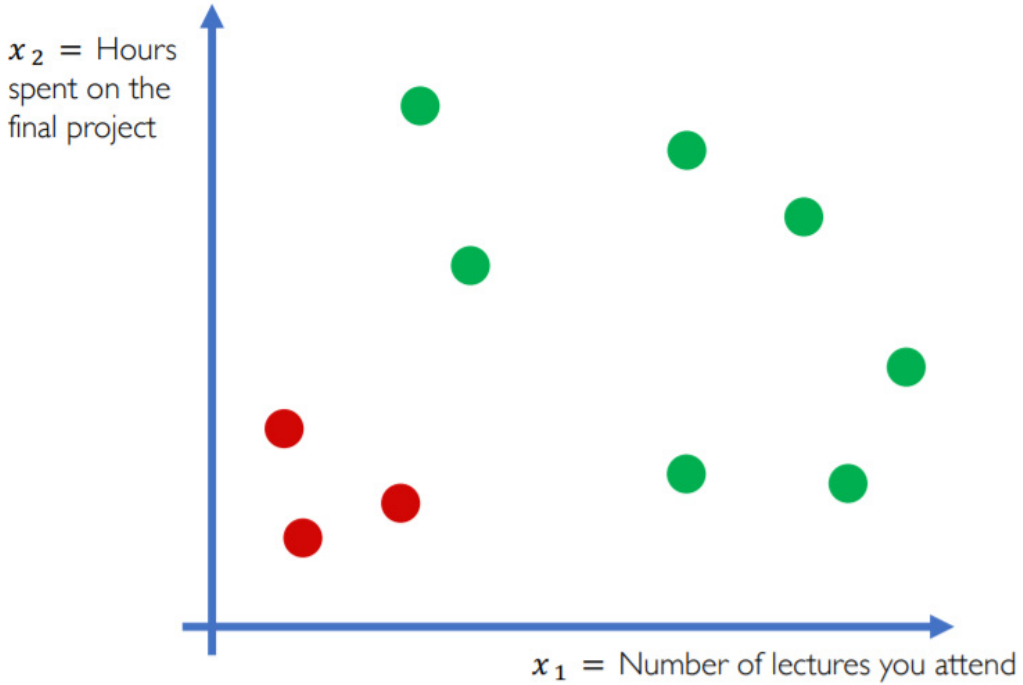


$\boldsymbol{X} \in \mathbb{R}^n \quad \rightarrow f(\boldsymbol{X};\boldsymbol{W}) \rightarrow \quad \boldsymbol{Y} \in \mathbb{R}^m$

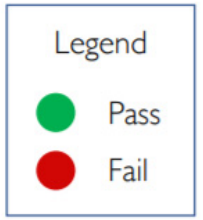
EXAMPLE PROBLEM

Will I pass this class?

- Train a neural network to determine if you will pass this class
- Simple two feature model
  - x_1= number of lectures you attend
  - x_2= hours spent on the final project



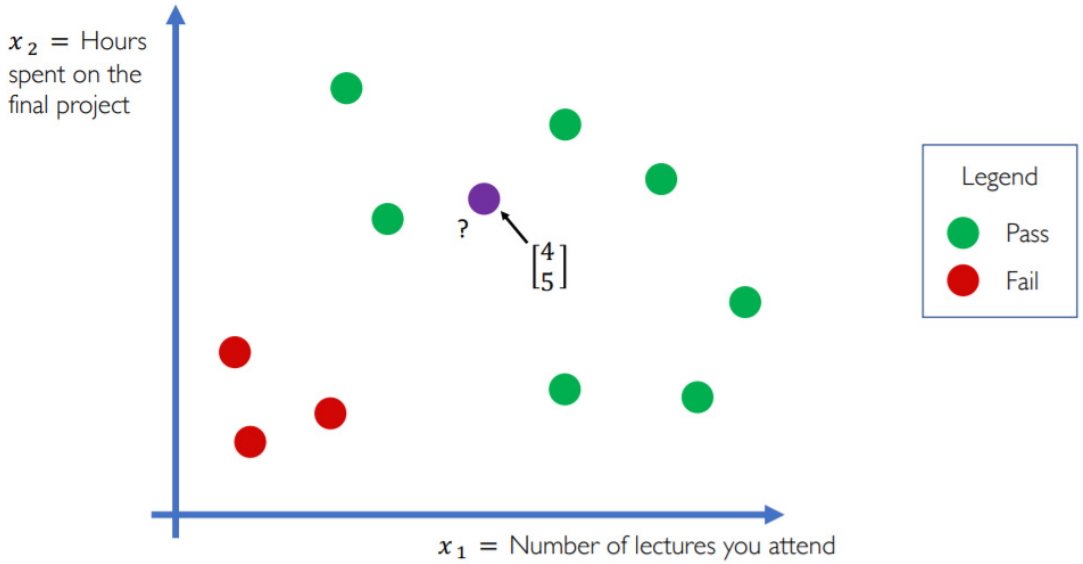
Annotated dataset
Previous students from previous years that



EXAMPLE PROBLEM

Will I pass this class?

- If you want to know if you are going to pass the class, you can plot yourself in the features space
  - E.g., 4 lectures attended, and 5 hours spent on the final project
  - Are you going to pass or fail?

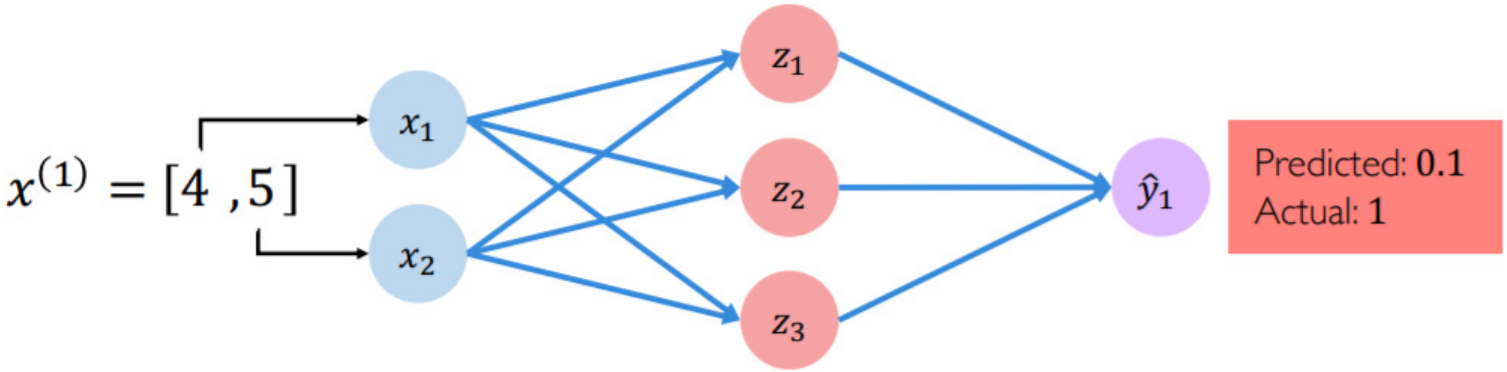


- **How to build a neural network to learn this by looking at the previous students (annotated dataset)?**

EXAMPLE PROBLEM

Will I pass this class?

- E.g., 2 inputs fed into a single layer neural network with 3 hidden units
- The final output probability that you will pass the exam is 0.1 (10%)
- This outcome contradicts the expectations when looking at the feature space
- Why the neural network provide such a bad prediction?
- The network **was not trained**
  - At this moment these are just meaningless input numbers

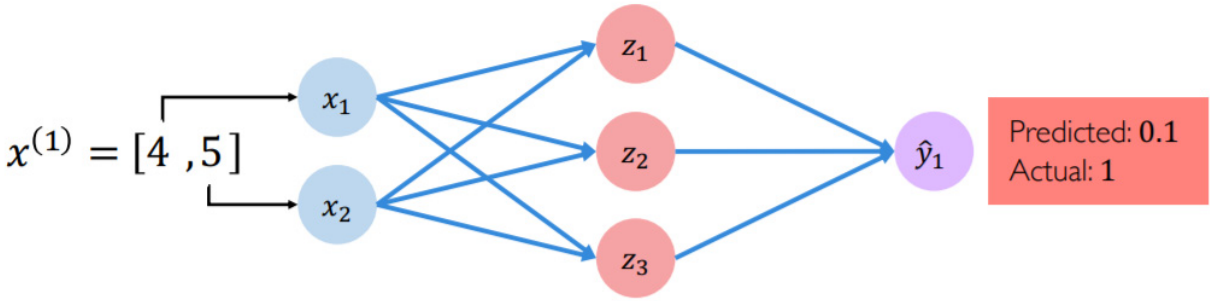


- Need to teach the network how to get the right answer

QUANTIFYING LOSS

How to tell the network when it makes a mistake

- The **loss** of the network:
  - Is what defines when the network makes the wrong prediction
  - Measures the cost incurred from incorrect predictions



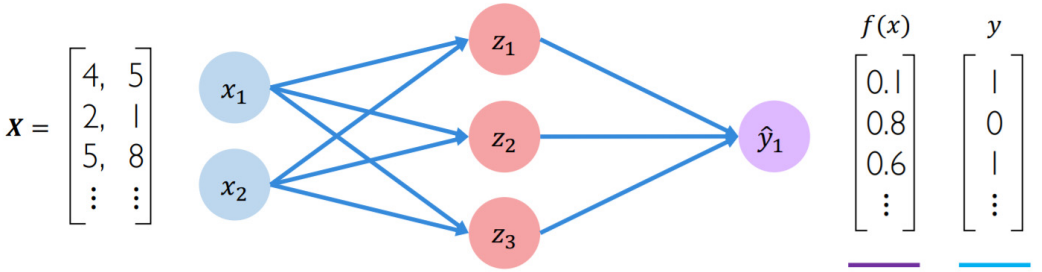
- It takes as input the predicted output and the groundtruth actual output
  - If they are close, the loss is going to be low

$$\mathcal{L}(f(x^{(i)}; \boldsymbol{W}), y^{(i)})$$

PREDICTED ACTUAL

EMPIRICAL LOSS

- **Input data**: from many students
- How the model perform across the entire population of students?
- The **empirical loss** measures the total loss over our entire dataset
  - Compute the loss of each of the student
  - And compute the mean
- When training the network we don't want to minimize the loss for a particular student but we want to minimize the loss across the entire dataset



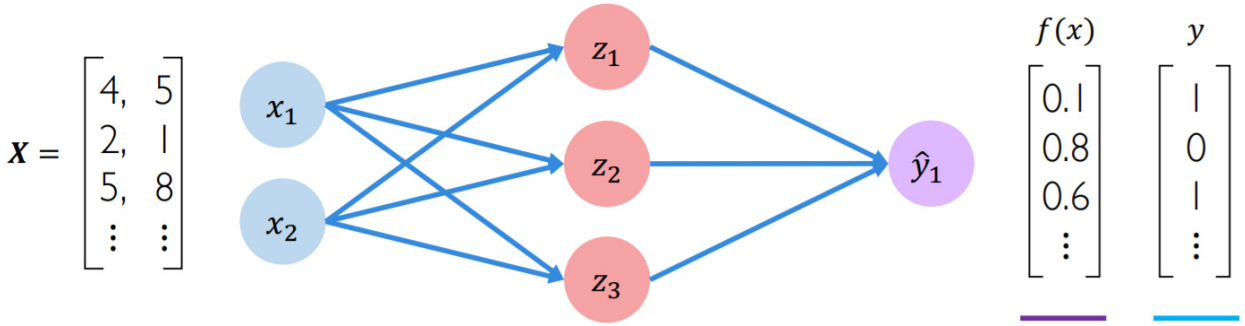
Also known as:
- Objective function
- Cost function
- Empirical Risk

$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i-1}^{n} \mathcal{L}(f(x^{(i)}; \boldsymbol{W}), y^{(i)})$$

PREDICTED ACTUAL

BINARY CROSS ENTROPY LOSS

- This is a binary classification problem (**soft max** can be used)
- **Cross entropy loss** can be used with models that output a probability between 0 and 1
  - Compute the loss between 0 and 1 outputs and the true labels



$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i-1}^{n} \underbrace{y^{(i)}}_{\text{ACTUAL}} \log(\underbrace{f\left(x^{(i)}; \boldsymbol{W}\right)}_{\text{PREDICTED}}) + \underbrace{\left(1 - y^{(i)}\right)}_{\text{ACTUAL}} \log(1 - \underbrace{f\left(x^{(i)}; \boldsymbol{W}\right)}_{\text{PREDICTED}})$$

LOSS OPTIMIZATION

- **How to use the loss to iteratively update and the train the network over time given some data?**

- **Objective**: find the network weights that minimize the **empirical loss**

$$\boldsymbol{W}^* = \arg\min_{\boldsymbol{W}} \frac{1}{n} \sum_{i-1}^{n} \mathcal{L}(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)})$$

$$\boldsymbol{W}^* = \arg\min_{\boldsymbol{W}} \mathrm{J}(\mathbf{W})$$

$\mathbf{W} = \{\mathbf{W}^{(0)}, \mathbf{W}^{(1)}, ...\}$ Set of all the weights in the network (first layer, second layer, etc..)

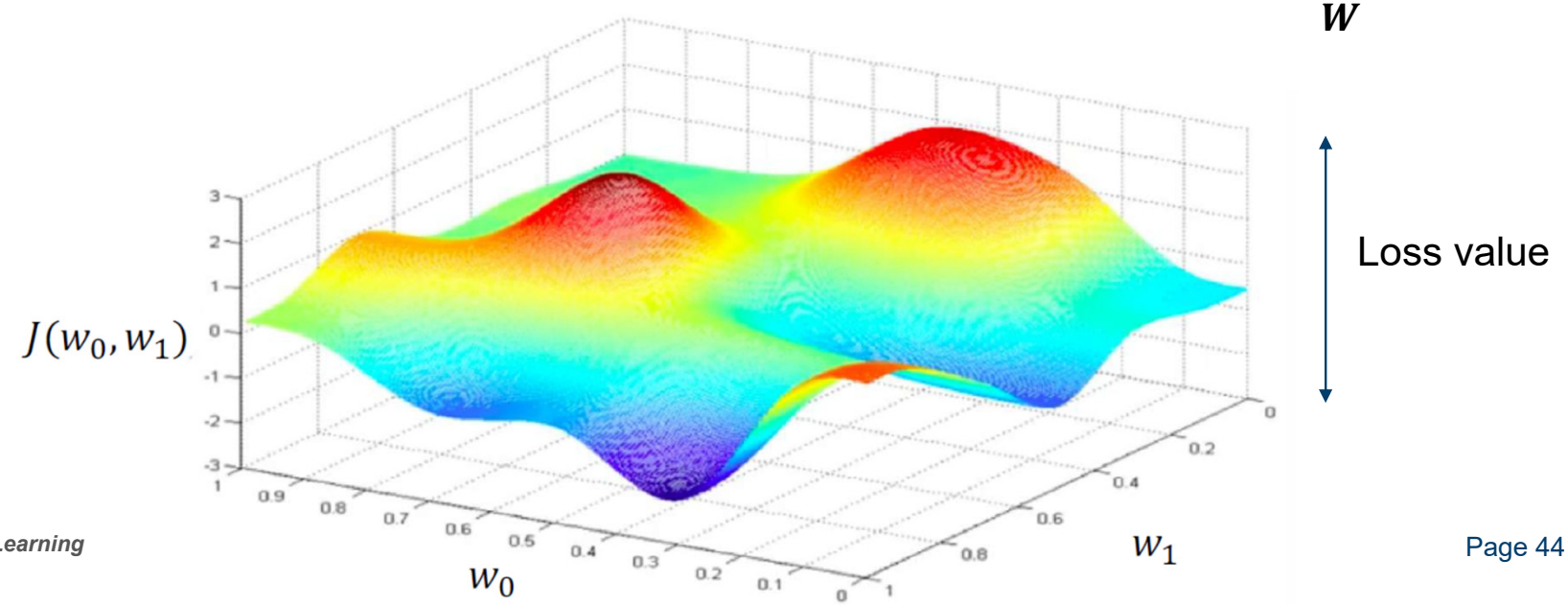
Compute this optimization problem over all these weights

LOSS OPTIMIZATION

What a loss function looks like

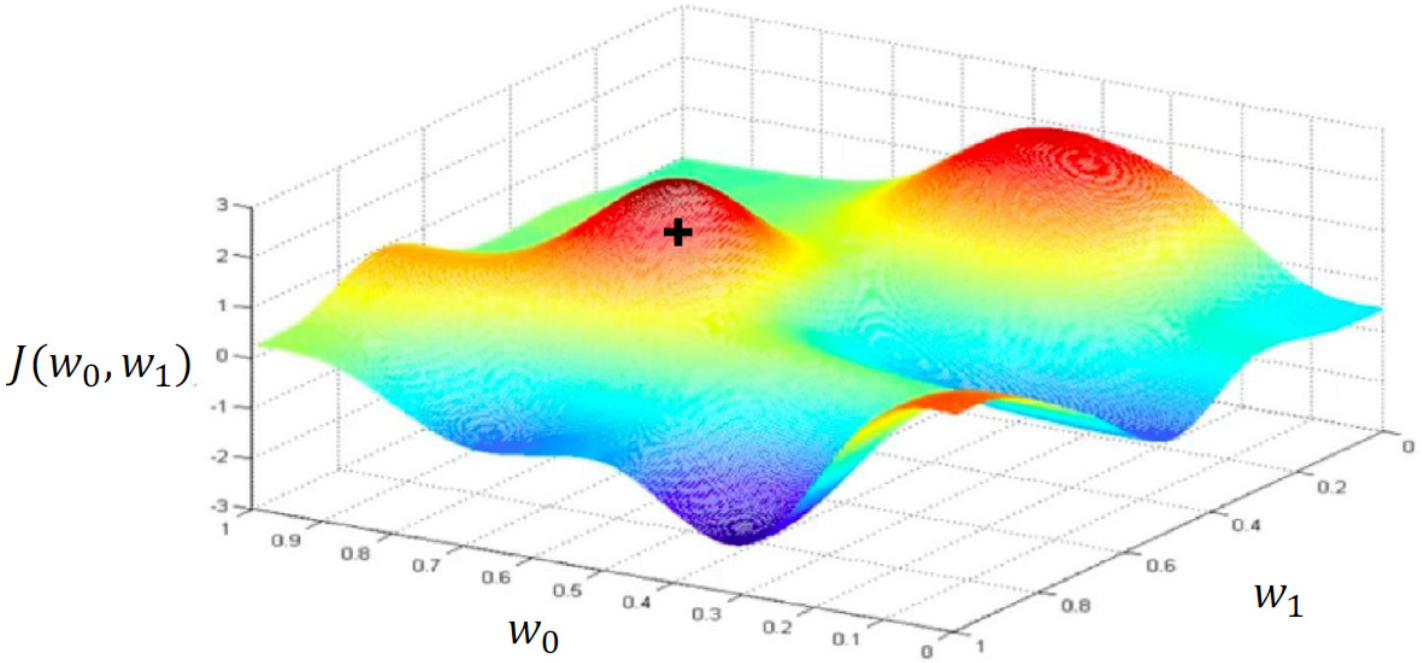
- Function that takes as input the weights and gives the loss
- The loss is a function of the network weights
- **Find the lowest point** in this landscape that correspond to the minimum loss
  - Find the correspondent weights

$$\boldsymbol{W}^* = \arg\min_{\boldsymbol{W}} J(\mathbf{W})$$



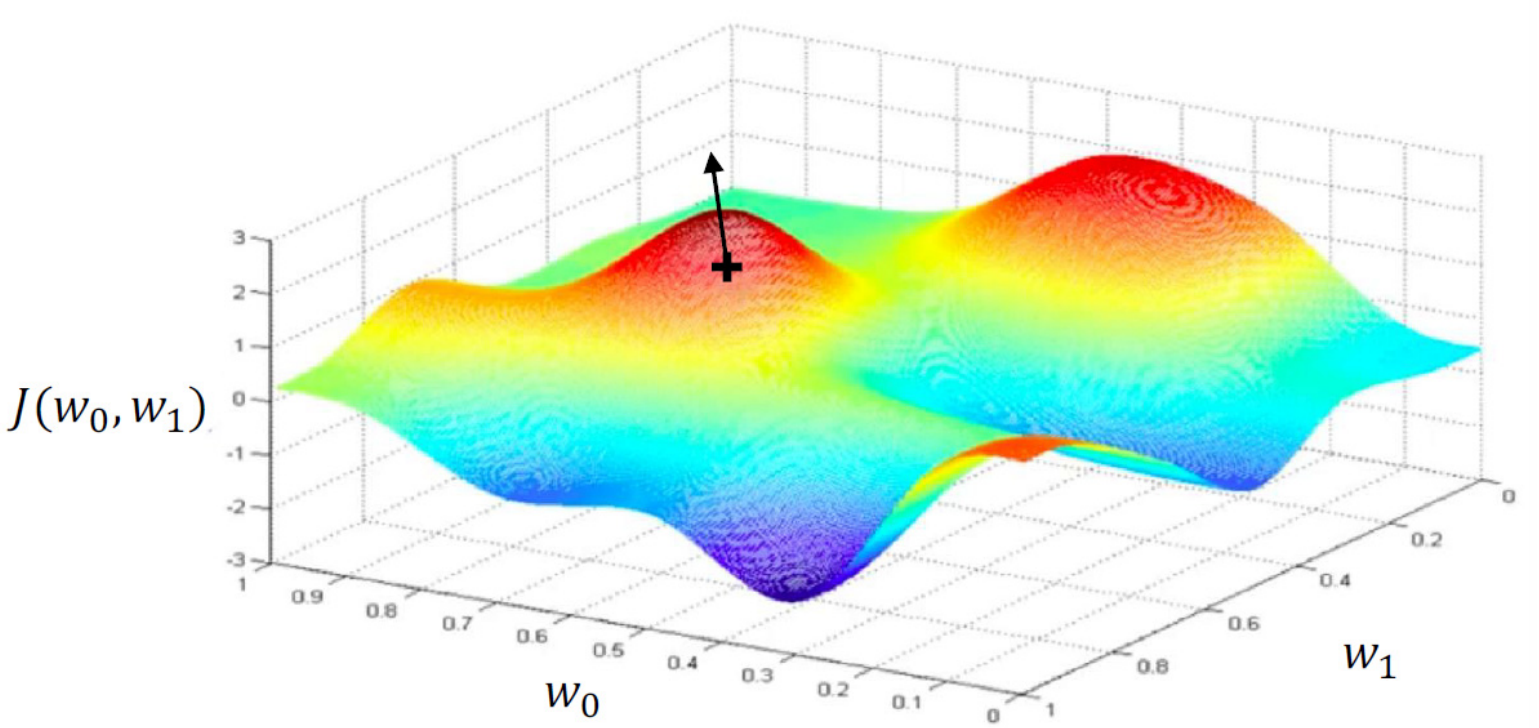
LOSS OPTIMIZATION

- Randomly pick an initial (w_0, w_1)



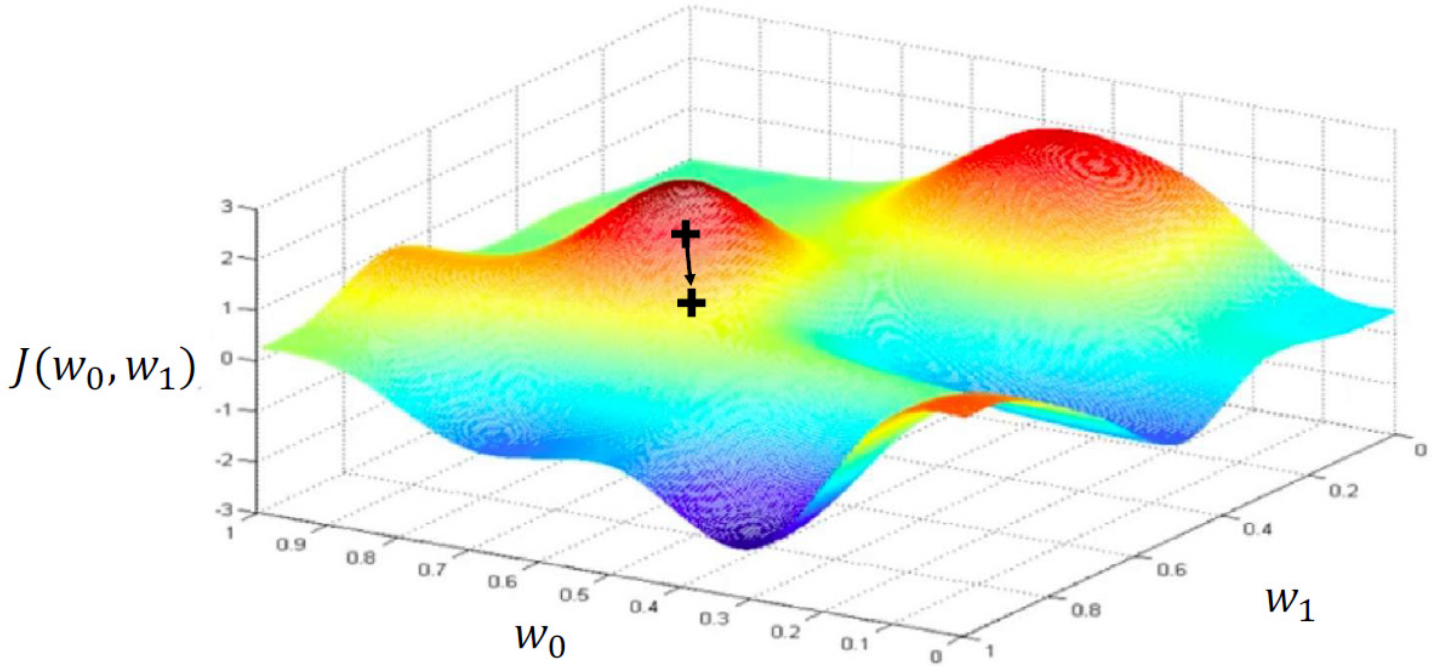
LOSS OPTIMIZATION

- Compute **gradient** at this local point $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
  - In this landscape the gradient tells us **the direction** of the maximum (steepest) **ascent**



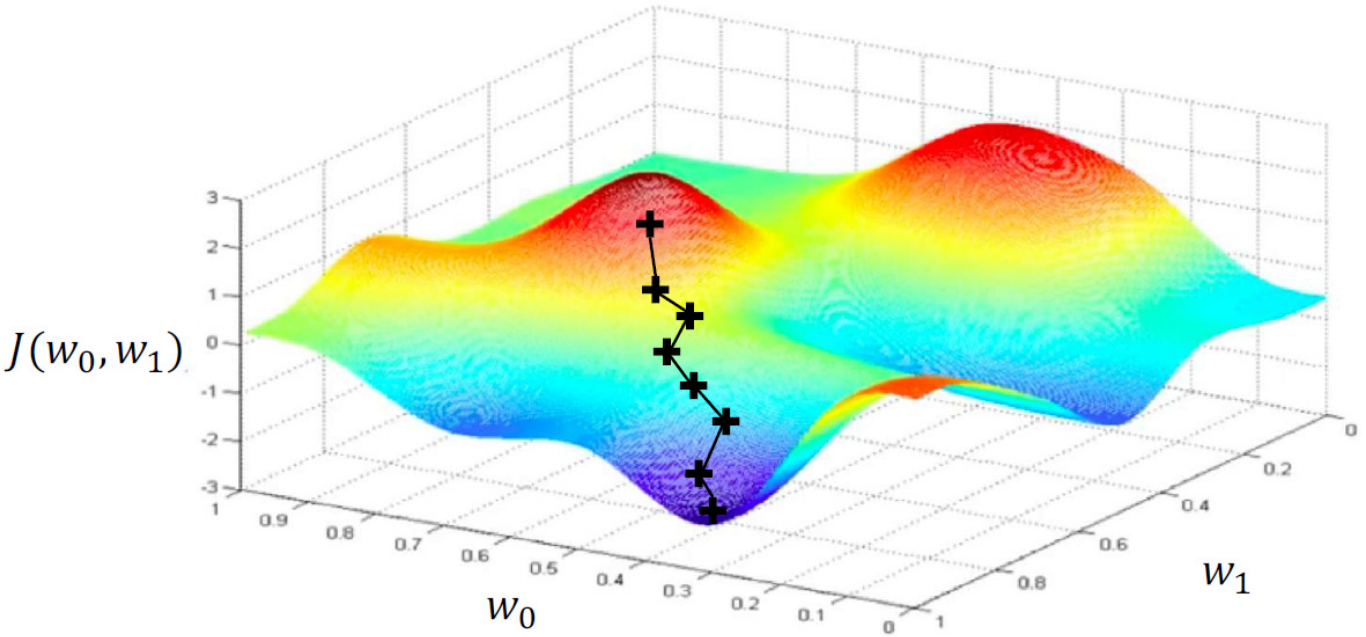
LOSS OPTIMIZATION

- Reverse the gradient and take a small step in opposite direction



GRADIENT DESCENT

- Repeat until convergence
  - The gradient is computed over and over



GRADIENT DESCENT

Algorithm

- How to compute it? This is a crucial part of deep learning and neural networks in general
- This is what all matters when optimizing the network
- It is also the most computational part

This procedure can be summarizing with the following algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence
3. Compute gradient $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$ (it explains how the loss changes with respect to each of the weights)
4. Update weights $W := W - \eta \frac{\partial J(\boldsymbol{W})}{\partial W}$ (in the opposite direction of the gradient)
5. Return weights

Use a small amount (the step) i.e., the **learning rate**

I.e., How much do you trust the computed gradient

LOSS FUNCTIONS

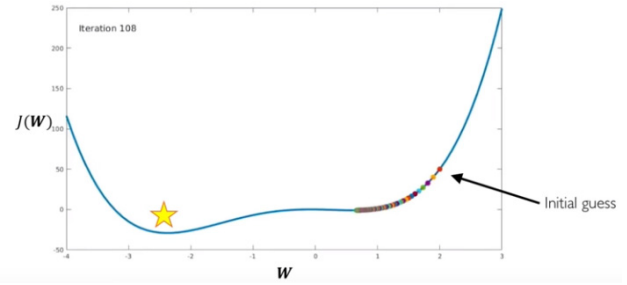
Optimization through gradient descent

- Take the weights and subtract, move via the negative gradient

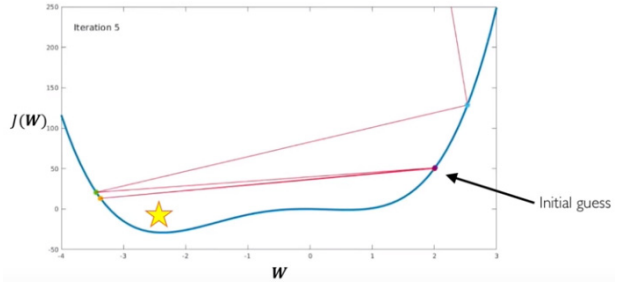
$$W := W - \eta \frac{\partial \mathcal{L}(W)}{\partial W}$$

How to set the **learning rate**?

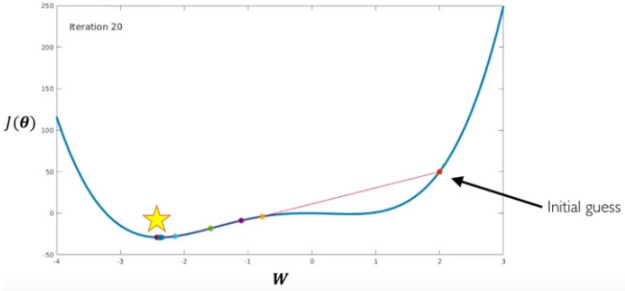
- How large is the step we take at each iteration
  - In practice it is very difficult to set this parameter
  - It is very important in order to avoid local minima



Small learning rate converges slowly and gets stuck in false local minima



Large learning rates overshoot, become unstable and diverge



Stable learning rates converge smoothly and avoid local minima

COMPUTING GRADIENTS: BACKPROPAGATION

Example with a simple network, one input, one hidden unit and one output

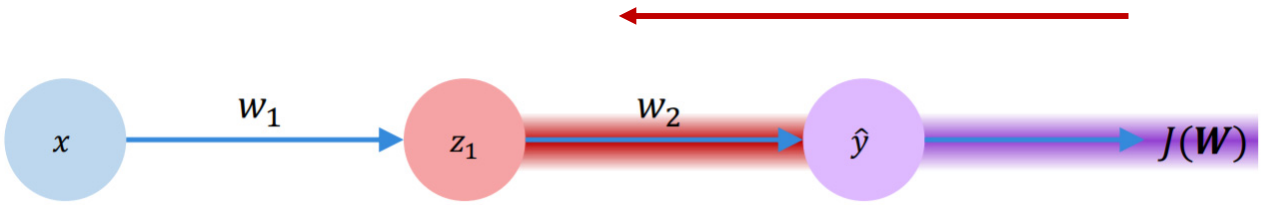
- Computing the gradient of the loss with the respect to $\boldsymbol{w_2}$ correspond to tells us how much a small change in $\boldsymbol{w_2}$ affects our loss
- How does a small change in one weight (e.g., $\boldsymbol{w_2}$) affect the final loss $J(\boldsymbol{W})$?

COMPUTING GRADIENTS: BACKPROPAGATION

- If it is written as a derivative
- Start by computing this by simply expanding this derivative by using the **chain rule**

$$\frac{\partial J(\boldsymbol{W})}{\partial w_2}$$ Use the chain rule

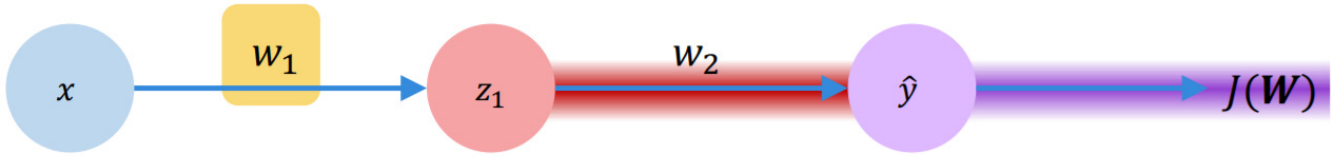
- **Backward from the loss trough the output**



$$\frac{\partial J(\boldsymbol{W})}{\partial w_2} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2}$$

COMPUTING GRADIENTS: BACKPROPAGATION

- And we compute for w_1 it looks like

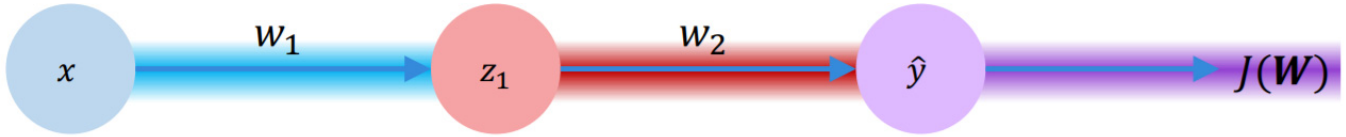


$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2}$$

apply chain rule

apply chain rule

- Back propagate the gradients further back to the network



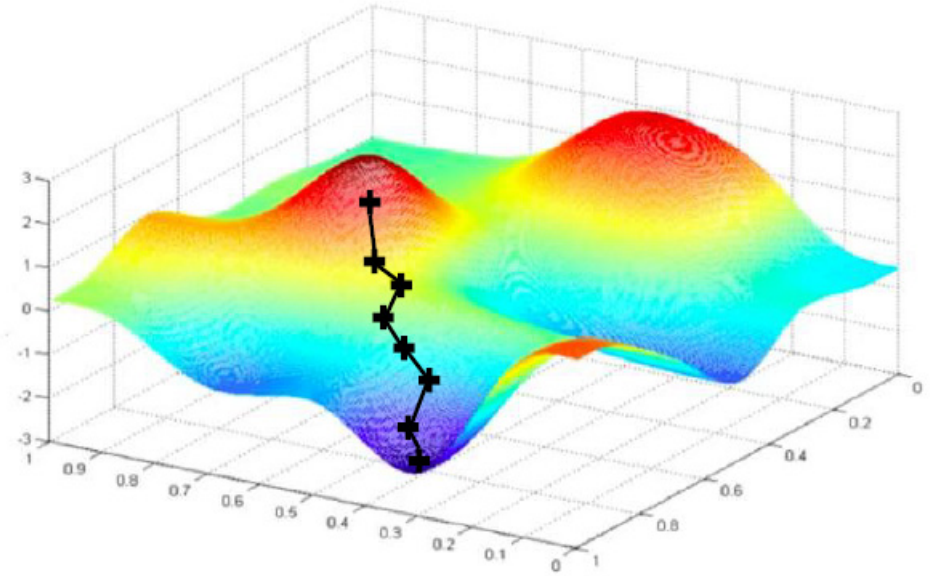
$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

- Repeat this **for every weight in the network**
  - Using gradients from later layers to backpropagate those errors back into the original input
- Do this for all the weights and **this gives us the gradient for each weight**

BATCH GRADIENT DESCENT (BGD)

Gradient is very computational to compute

- When it is computed for all the samples within a given training dataset

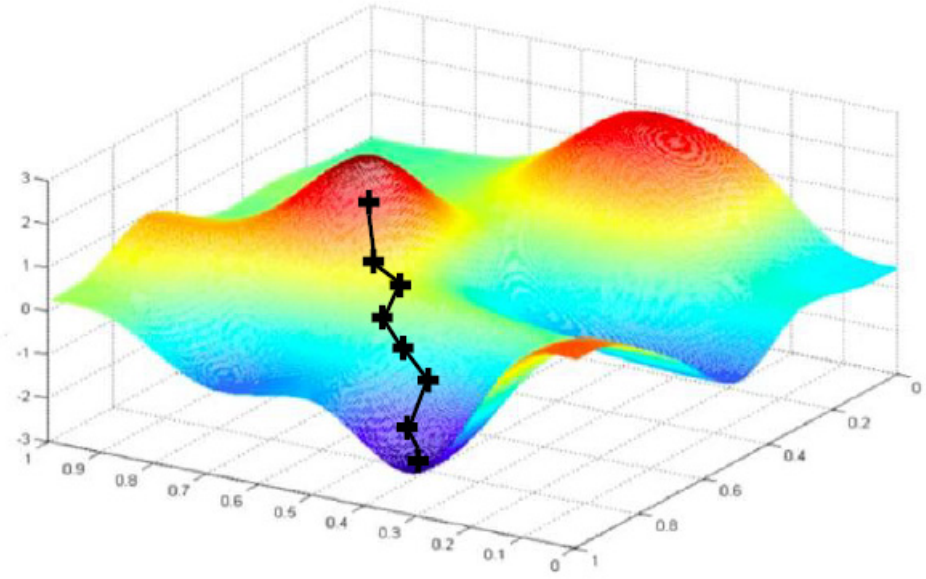


1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence
3. Compute gradient $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights $W := W - \eta \frac{\partial \mathcal{L}(X;W;Y)}{\partial W}$
5. Return weights

STOCHASTIC GRADIENT DESCENT (SGD)

- Strategy
  - Pick a single point
  - Compute the gradient with respect to that single point and use it to update the weights
- We might go in a direction/step that is not representative for the entire dataset

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence
3. **Pick single data point i**
4. Compute gradient $\frac{\partial J_i(W)}{\partial W}$ ← **Easy to compute but very noisy (stochastic)!**
5. Update weights $W := W - \eta \frac{\partial \mathcal{L}(X;W;Y)}{\partial W}$
6. Return weights

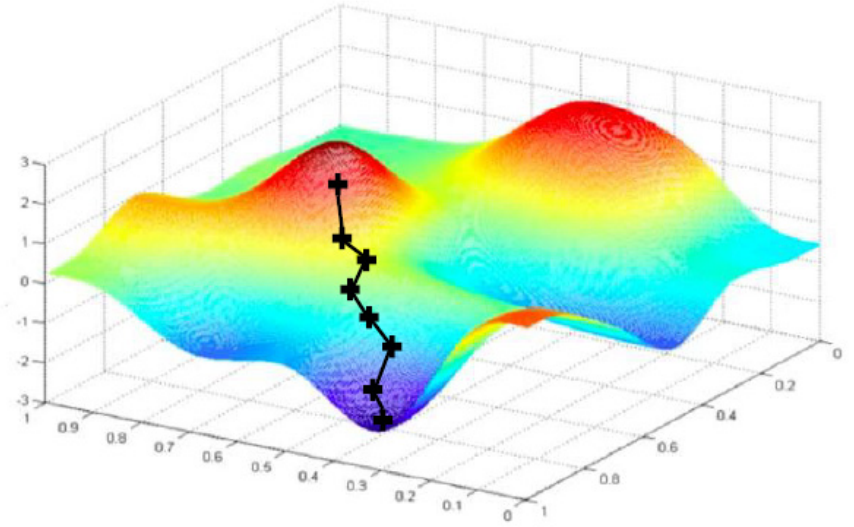


MINI-BATCH GRADIENT DESCENT

Batching the data into mini-batches

- Tradeoff between **BGD** and **SGD**
- Gives an estimate of the true gradient by averaging the gradient from each of the B points
- Minibatch sampling: implemented by shuffling the dataset S, and processing that permutation by obtaining contiguous segments of size B from it

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence
3. **Pick batch of B data points**
4. Compute gradient $\frac{\partial J_i(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{\mathbf{1}}{\boldsymbol{B}} \sum_{k=1}^{B} \frac{\partial J_k(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update weights $W := W - \eta \frac{\partial \mathcal{L}(X;W;Y)}{\partial W}$
6. Return weights



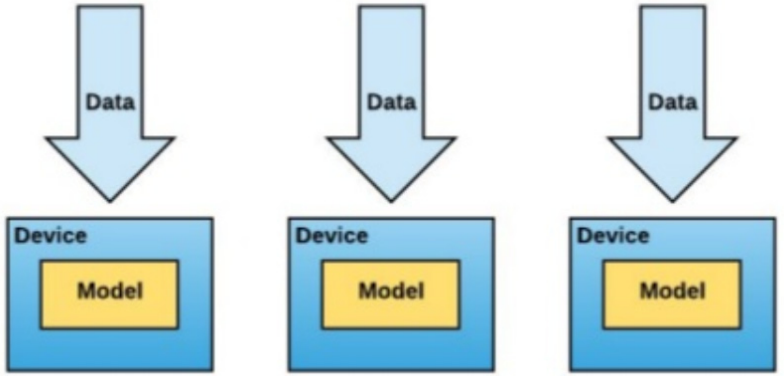
Fast to compute and a much better estimate of the true gradient!

DISTRIBUTED TRAINING

With Data Parallelism

- **Mini-Batch Gradient Descent:**
  - More accurate estimation of gradient and smoother convergence
  - Allows for larger learning rates (i.e., trust more the gradient , training faster)
  - Can **parallelize computation** and achieve significant speed increases

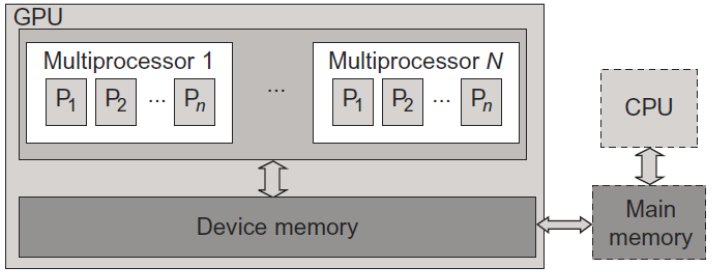
Send batches across the GPUs, compute their gradient simultaneously and aggregate them back



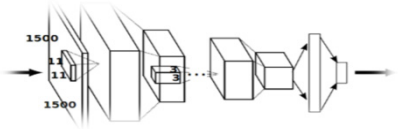
[17] Horovod

GPU Acceleration (cf. Lecture 1)

- GPU accelerator architecture example (e.g. NVIDIA card)
  - GPUs can have 128 cores on one single GPU chip
  - Each core can work with eight threads of instructions
  - GPU is able to concurrently execute 128 * 8 = 1024 threads
  - Interaction and thus major (bandwidth) bottleneck between CPU and GPU is via memory interactions
  - E.g. applications that use matrix – vector/matrix multiplication (e.g. deep learning algorithms)



[18] Distributed & Cloud Computing Book

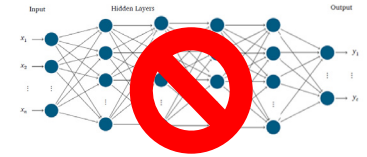


- **CPU acceleration means that GPUs accelerate computing due to a massive parallelism with thousands of threads compared to only a few threads used by conventional CPUs**
- **GPUs are designed to compute large numbers of floating point operations in parallel**

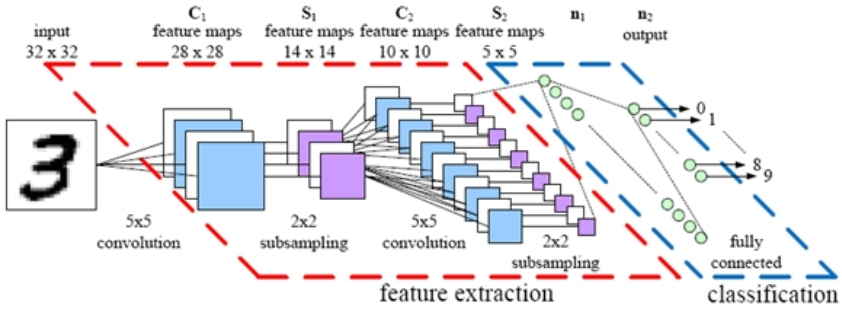
➢ **Lecture 10 will introduce the programming of accelerators with different approaches and their key benefits for applications**

DEEP Learning takes advantage of Many-Core Technologies (cf. Lecture 1)

[20] Neural Network 3D Simulation



- **Innovation via specific layers and architecture types**



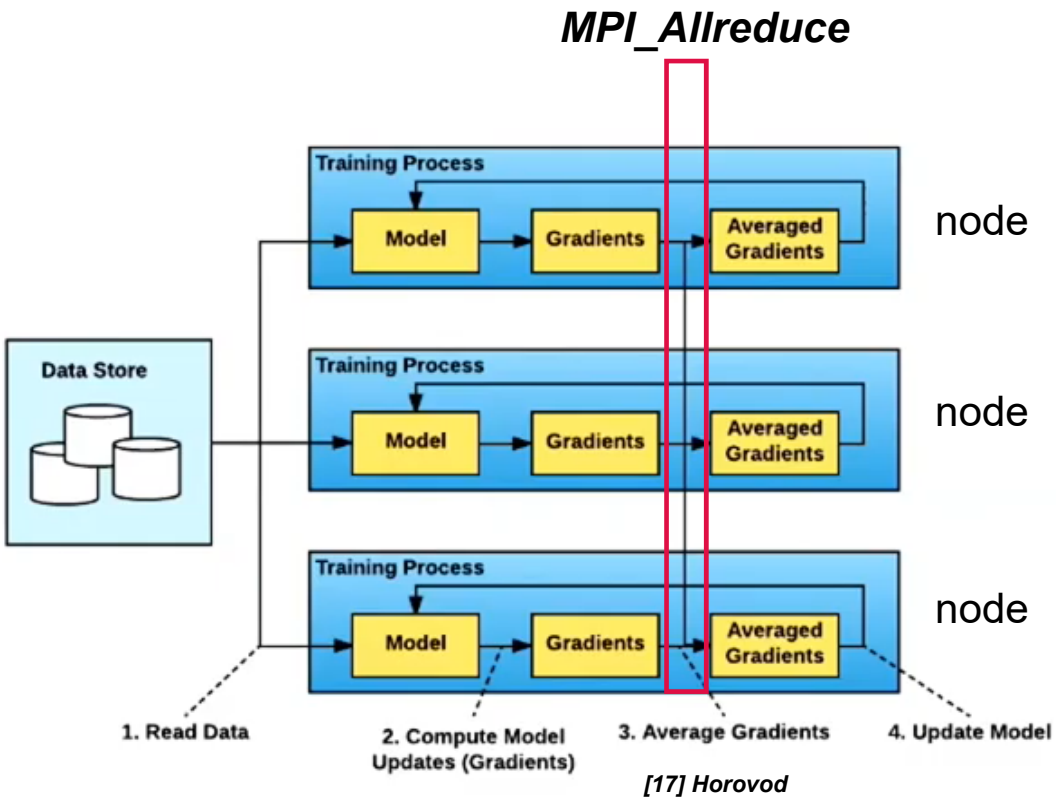
[21] A. Rosebrock

➢ **Lecture 8 will provide more details about parallel & scalable machine & deep learning algorithms and how many-core HPC is used**

DISTRIBUTED TRAINING

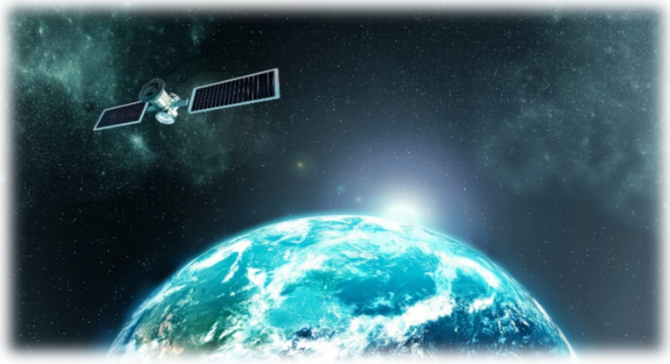
With Data Parallelism

- The gradients for different batches of data are calculated separately on each node
- But averaged across nodes to apply consistent updates to the model copy in each node



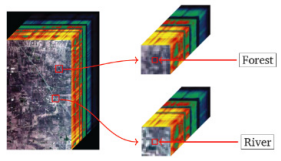
[17] Horovod

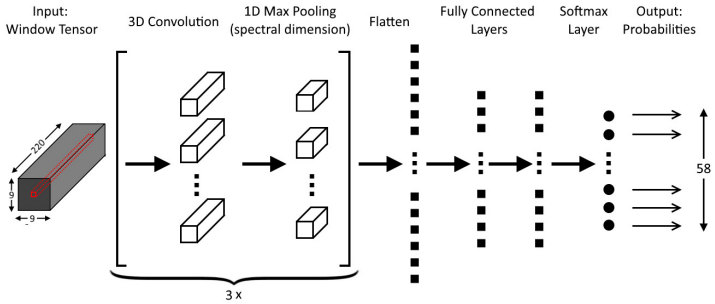
Deep Learning Application Example – Using HPC (cf. Lecture 1)



- **Using Convolutional Neural Networks (CNNs) with hyperspectral remote sensing image data**

[22] J. Lange and M. Riedel et al., IGARSS Conference, 2018

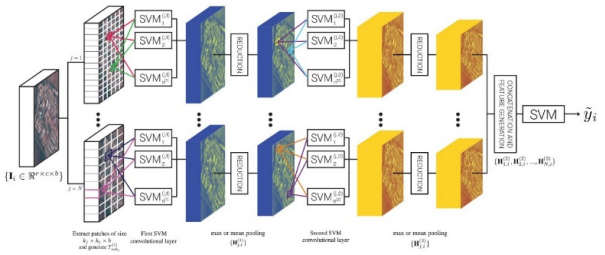




Feature	Representation / Value
Conv. Layer Filters	48, 32, 32
Conv. Layer Filter size	(3, 3, 5), (3, 3, 5), (3, 3, 5)
Dense Layer Neurons	128, 128
Optimizer	SGD
Loss Function	mean squared error
Activation Functions	ReLU
Training Epochs	600
Batch Size	50
Learning Rate	1
Learning Rate Decay	5×10^{-6}

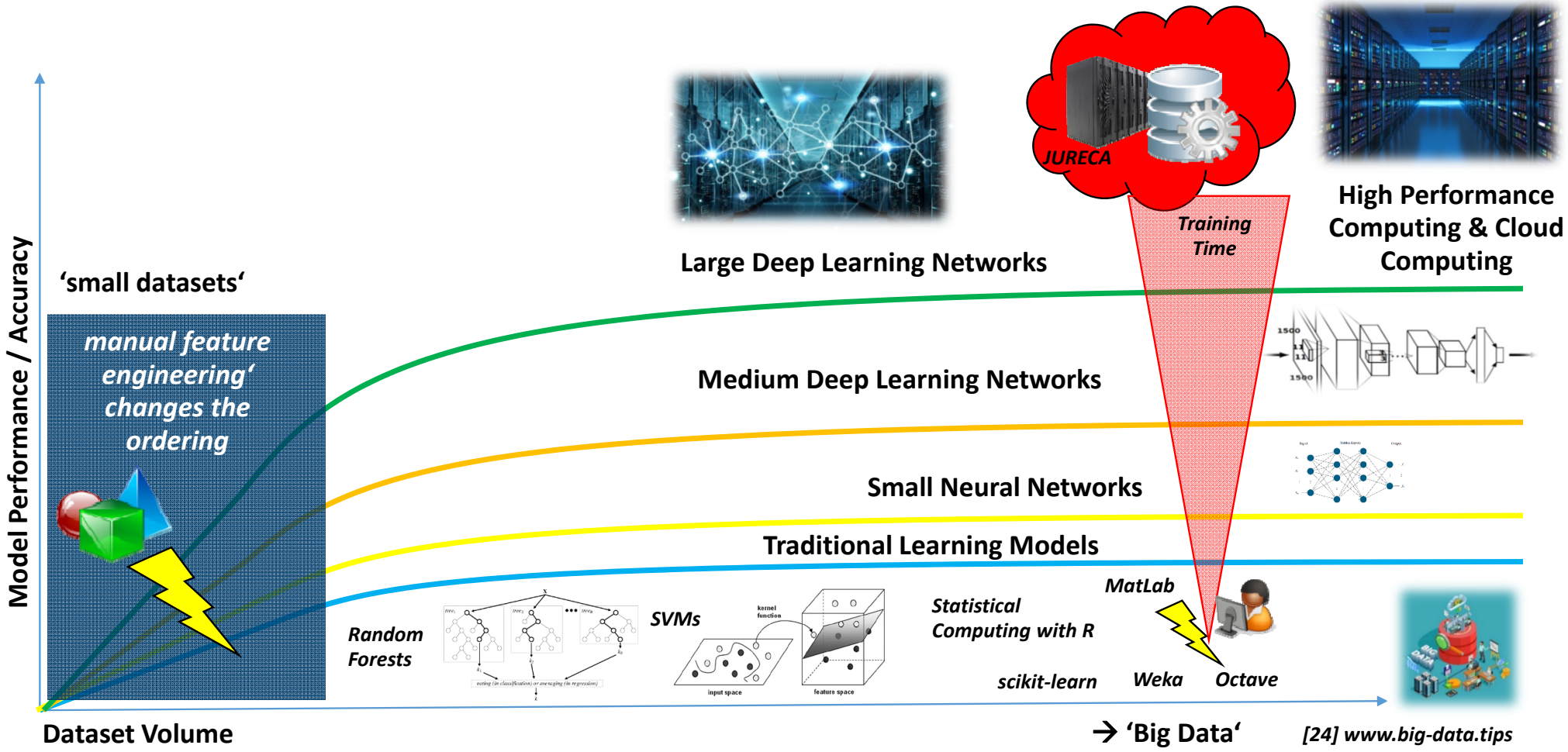
[23] G. Cavallaro, M. Riedel et al., IGARSS 2019

- **Find Hyperparameters & joint 'new-old' modeling & transfer learning given rare labeled/annotated data in science (e.g. 36,000 vs. 14,197,122 images ImageNet)**



➢ **Lecture 8 will provide more details about parallel & scalable machine & deep learning algorithms and remote sensing applications**

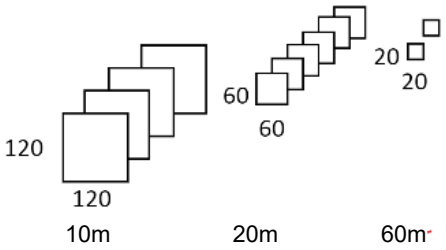
HPC Relationship to ‘Big Data‘ in Machine & Deep Learning (cf. Lecture 1)

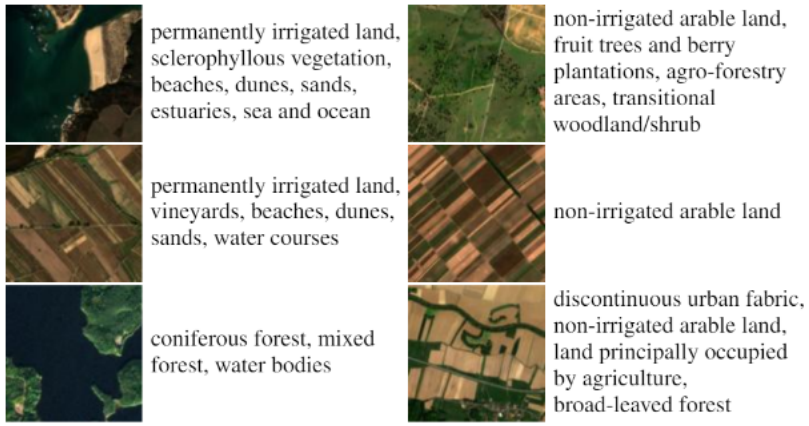


EXAMPLE

Multispectral Remote Sensing Dataset

Datasets	Image type	Image per class	Scene classes	Annotation type	Total images	Spatial resolution (m)	Image sizes	Year	Ref. [9] G. Sumbul et al.
BigEarthNet	Satellite MS	328 to 217119	43	Multi label	590,326	10 20 60	120x120 60x60 20x20	2018	





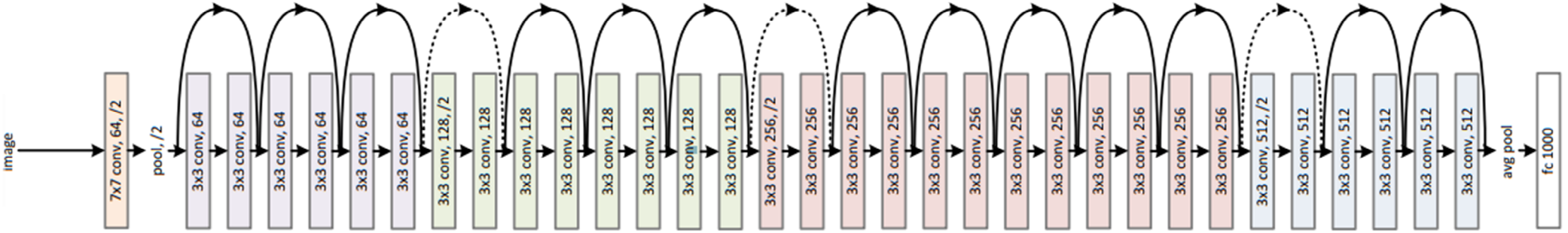
https://www.tensorflow.org/datasets/datasets#bigearthnet

[9] G. Sumbul et al.

RESNET-50

Classifier

- **25.6 millions of trainable parameters**

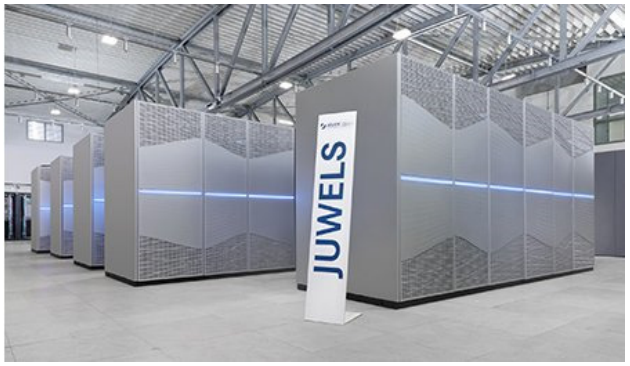


- It has established a strong baseline in terms of accuracy
  - Representing good trade-off between accuracy, depth and number of parameters
- It is very well suitable for parallelization: **distributed training**

EXPERIMENTS

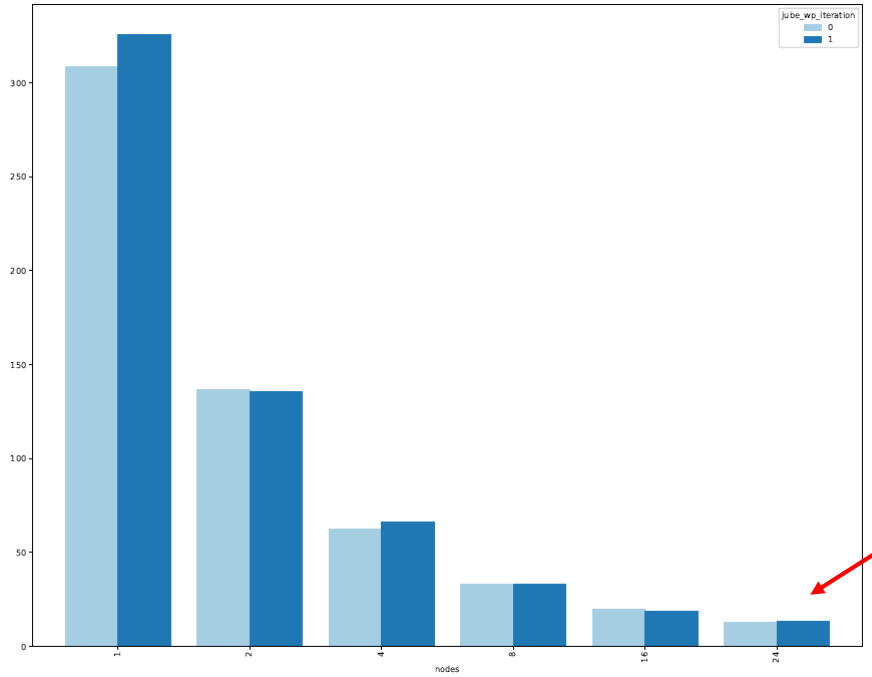
At JSC with the JUWELS HPC System

- A partition of the system consists of 56 compute nodes
  - With each having four Nvidia V100 GPUs (equipped with 16GB of memory)



Jülich Wizard for European Leadership Science (JUWELS)

Time per epoch [sec]



24 nodes x 4 GPUs = 96 GPUs

REFERENCES (1)

- [1] CORONA: American's First Satellite Program: first photograph.
  Online: https://www.oneonta.edu/faculty/baumanpr/geosat2/RS%20History%20II/RS-History-Part-2.html
- [2] The Earth-Atmosphere Energy Balance
  Online: http://theatmosphere.pbworks.com/w/page/27058542/The%20Earth-Atmosphere%20Energy%20Balance
- [3] Electromagnetic Radiation and the Electromagnetic Spectrum
  Online: https://remotesensing9302006.wordpress.com/2013/12/26/electromagnetic-radiation-and-the-electromagnetic-spectrum/
- [4] K. Tempfli, N. Kerle, G. C. Huurneman, L. L. F. Janssen, Principles of Remote Sensing: An Introductory TextBook
  Online: https://webapps.itc.utwente.nl/librarywww/papers_2009/general/principlesremotesensing.pdf
- [5] AVIRIS Concept
  Online: https://aviris.jpl.nasa.gov/aviris/concept.html
- [6] Paris, C., Bruzzone, L., & Fernandez-Prieto, D. (2019). A Novel Approach to the Unsupervised Update of Land-Cover Maps by Classification of Time Series of Multispectral Images. IEEE Transactions on Geoscience and Remote Sensing. https://doi.org/10.1109/TGRS.2018.2890404
- [7] Satellite Images Visual Interpretation Online Training Course
  Online: https://www.youtube.com/watch?v=eDND1sGMqF8
- [8] P. Helber, B. Bischke, A. Dengel, and D. Borth, "EuroSAT: A Novel Dataset and Deep Learning Benchmark for Land Use and Land Cover Classification," Computing Research Repository - arXiv, vol. abs/1709.0, 2017
- [9] G. Sumbul, M. Charfuelan, B. Demir, V. Markl, BigEarthNet: A Large-Scale Benchmark Archive for Remote Sensing Image Understanding, IEEE International Conference on Geoscience and Remote Sensing Symposium, Yokohama, Japan, 2019.
- [10] Remote Sensing for All Sectors
  Online: https://www.zu.ac.ae/main/en/colleges/colleges/__college_of_natural_and_health_sciences/research/_hyperspectral/_pages/Initiatives.aspx

REFERENCES (2)

- [11] Sentinel-2 Satellite Imagery
  Online: http://geocento.com/satellite-imagery-gallery/sentinel-2/
- [12] CORINE Land Cover — Copernicus Land Monitoring Service
  Online: https://land.copernicus.eu/pan-european/corine-land-cover
- [13] F. Ömrüuzun, B. Demir, L. Bruzzone and Y. Y. Çetin, "Content based hyperspectral image retrieval using bag of endmembers image descriptors," 8th Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS), Los Angeles, CA, 2016, pp. 1-4.
- [14] Convolutional Neural Networks
  Online: https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/convolutional_neural_networks.html
- [15] CS231n Convolutional Neural Networks for Visual Recognition
  Online: http://cs231n.github.io/
- [16] A. Krizhevsky, I.Sutskever and G. E. Hinton, “ImageNet classification with deep convolutional neural networks”, in *Communications of the ACM*, vol. 60 Issue 6, pp. 84-90, 2017.
- [17] Horovod: Uber’s Open Source Distributed Deep Learning Framework for TensorFlow
  Online: https://www.slideshare.net/databricks/horovod-ubers-open-source-distributed-deep-learning-framework-for-tensorflow
- [18] K. Hwang, G. C. Fox, J. J. Dongarra, ‘Distributed and Cloud Computing’, Book, Online:
  Online:http://store.elsevier.com/product.jsp?locale=en_EU&isbn=9780128002049
- [19] DEEP Projects Web page, Online:
  Online:http://www.deep-projects.eu/
- [20] YouTube Video, ‘Neural Network 3D Simulation‘, Online:
  Online:https://www.youtube.com/watch?v=3JQ3hYko51Y

REFERENCES (3)

- [21] A. Rosebrock, ‘Get off the deep learning bandwagon and get some perspective‘, Online: http://www.pyimagesearch.com/2014/06/09/get-deep-learning-bandwagon-get-perspective/
- [22] J. Lange, G. Cavallaro, M. Goetz, E. Erlingsson, M. Riedel, ‘The Influence of Sampling Methods on Pixel-Wise Hyperspectral Image Classification with 3D Convolutional Neural Networks’, Proceedings of the IGARSS 2018 Conference
- [23] G. Cavallaro, Y. Bazi, F. Melgani, M. Riedel, ‘Multi-Scale Convolutional SVM Networks for Multi-Class Classification Problems of Remote Sensing Images’, Proceedings of the IGARSS 2019 Conference, to appear
- [24] Big Data Mining & Machine Learning

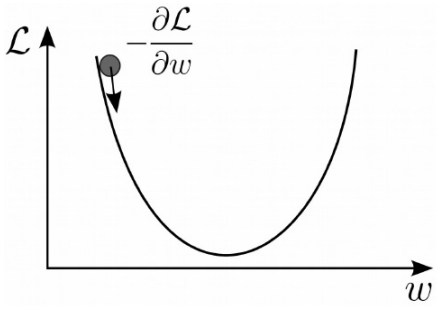
  Online: http://www.big-data.tips/

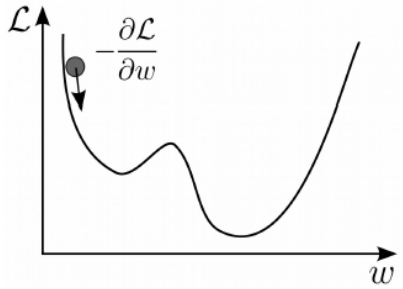
APPENDIX

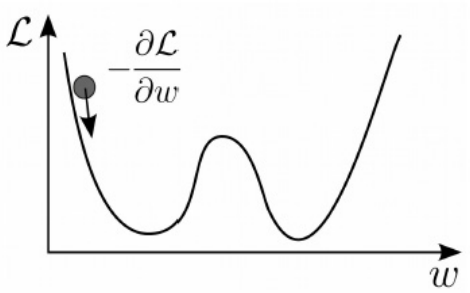
TRAINING DEEP NEURAL NETWORKS

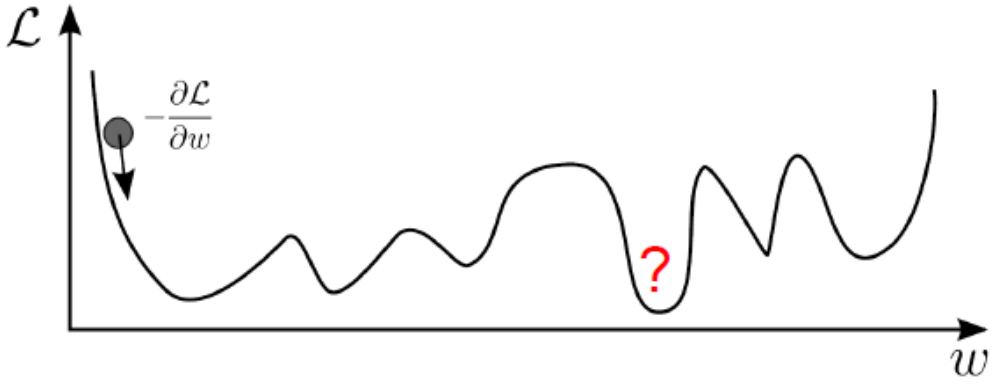
Loss minimization is highly non-trivial

- How to ensure to get to a global minima (instead of a local minima)?
  - There is no guarantee
- Also, many local minima
- Finding the optimal true minimum is difficult







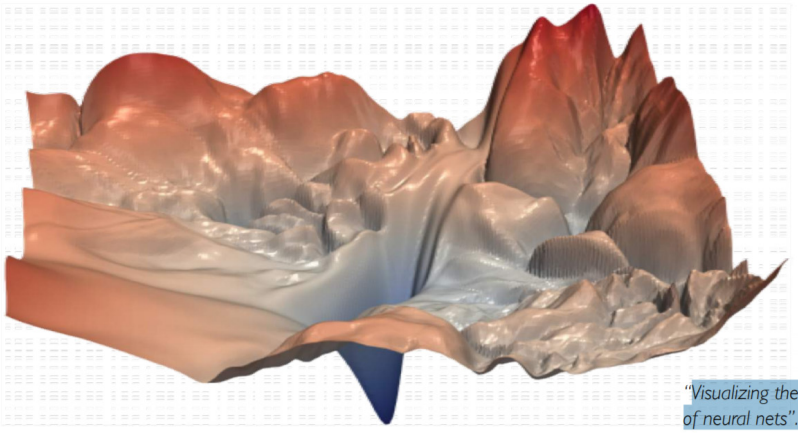


TRAINING DEEP NEURAL NETWORKS

Loss minimization is highly non-trivial

- There is one fundamental problem of machine learning
  - **The real/true loss function is hidden**
- You always work with the limited training data
  - No matter how huge it is
  - You are always working with a tiny subset of the real problem
- We compute the **empirical loss** (the loss on the training data)

- How to estimate the **true loss**? ,
- i.e., How good or bad the network performs on the data that the network did not see during the training?

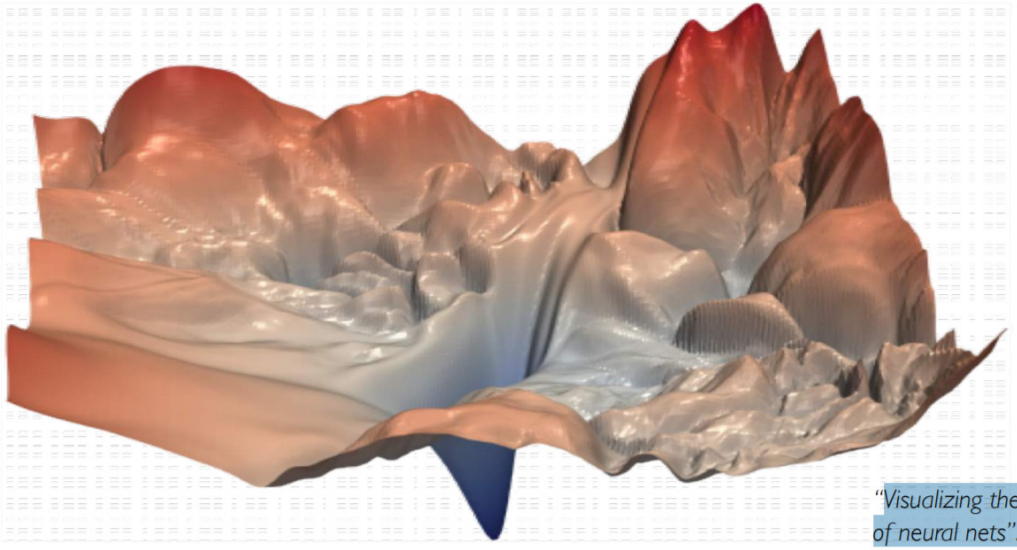


"Visualizing the loss landscape of neural nets". Dec 2017.

TRAINING DEEP NEURAL NETWORKS

Is it possible?

- One would expect that ML would never work (i.e., this kind of learning will just fail)
  - E.G., **Overfitting**, adapting too much to the training data (lookup table procedure - memorize)
- Previous common wisdom: learning in deep architectures not tractable



"Visualizing the loss landscape of neural nets". Dec 2017.

TRAINING DEEP NEURAL NETWORKS – KEY IMPROVEMENTS

What are the reason for the boost in performance

- Improved **stochastic gradient descent (SGD)**
  - Preventing vanishing or exploding gradients over many layers
    - Proper random **weight initialization**
    - **Suitable transfer functions**
    - Network architecture modification (e.g, **skip connections**)
  - Update rules with adaptive momentum (Nesterov), **adaptive learning rate** (stochastic annealing; Adam, RMSProp)
- **Regularization** against overfitting (DropOut, Batch Normalization, decaying weights, sparse activation, etc)
- **Huge amounts of labeled data** (ImageNet, CoCo, etc), freely available
- **Data augmentation** techniques extending data sets
- **Parallelization**, specialized hardware (e.g, GPU, TPU) based acceleration