



Machine Learning & Knowledge Extraction

DR KONSTANTINOS KARAMPIDIS

Πληροφορίες Μαθήματος

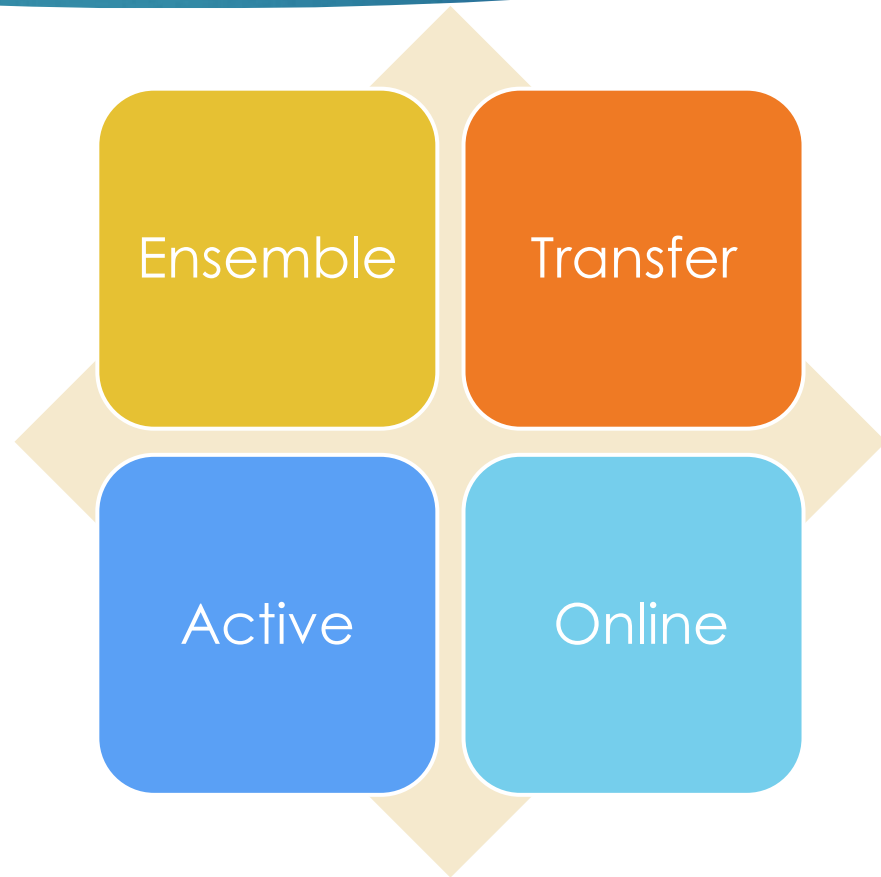
- ▶ Ωράριο:
 - ▶ Θεωρία: **Κάθε Τρίτη 09:00-13:00 – Αίθουσα 207**
 - ▶ Εργαστήριο: **13:00-14:00 ΕΡΓ6 (σύμφωνα με το πρόγραμμα που είναι αναρτημένο στο eclass)**
- ▶ Εργασίες
 - ▶ **1 project – Ομάδες έως 2 ατόμων – 80%**
 - ▶ **Εργαστηριακές ασκήσεις – 20%**
- ▶ Προαπαιτούμενα: Κανένα

Περιεχόμενο Μαθήματος

- ▶ Εισαγωγή στη Μηχανική Μάθηση - τι είναι, γιατί μας ενδιαφέρει, παραδείγματα προβλημάτων, η μηχανική μάθηση ως αναζήτηση, υπόθεση επαγωγικής μάθησης
- ▶ Επεξεργασία εισόδου – Μείωση διαστατικότητας- Αξιολόγηση
- ▶ Μέθοδοι επιβλεπόμενης μάθησης
- ▶ Νευρωνικά Δίκτυα
- ▶ Εξελικτική Μάθηση – Γενετικοί Αλγόριθμοι
- ▶ Μηχανική Μάθηση Βασιζόμενη σε Κανόνες
- ▶ Ενισχυτική Μάθηση
- ▶ **Μάθηση Αναπαράστασης**
- ▶ Εξόρυξη Δεδομένων

Other Types of Learning

Other
Types of
Learning



Ensemble

Ensemble learning is a machine learning method that integrates two or more machine learning models to create an ideal predictive model.

There are three basic strategies for ensemble learning:

- Bagging
- Stacking
- Boosting

Active

Active learning is a method where the machine learning model can question a human operator during the learning process to find an answer to any present vagueness during the learning cycle.

Active learning is a semi-supervised learning technique to achieve similar or improved results to traditional supervised learning while maintaining fewer training data.

Online

Online learning is another method that combines multiple machine learning techniques to stay updated with the latest data. Online learning utilizes available data and constantly updates the model before making a prediction or after the latest observation.

Online machine learning is specifically beneficial when the number of observations exceeds the memory limit. Hence, the learning is carried out incrementally over observations. In other words, the online learning model is continuously updated using a real-time data stream.

Limitations of Neural Networks

Random initialization + densely connected networks lead to:

- ▶ High cost
 - ▶ Each neuron in the neural network can be considered as a logistic regression.
 - ▶ Training the entire neural network is to train all the interconnected logistic regressions.
- ▶ Difficult to train as the number of hidden layers increases
 - ▶ Recall that logistic regression is trained by gradient descent.
 - ▶ In backpropagation, gradient is progressively getting more dilute. That is, below top layers, the correction signal δ_n is minimal.
- ▶ Stuck in local optima
 - ▶ The objective function of the neural network is usually not convex.
 - ▶ The random initialization does not guarantee starting from the proximity of global optima.
- ▶ Solution:
 - ▶ Deep Learning/Learning multiple levels of representation

Don't Forget !!!

- ▶ Logistic Regression (LR) is classified under Machine Learning (ML). One should not use Deep Learning (DL) for every possible problem.
- ▶ Don't use Machine Learning (ML) or Deep Learning (DL) if you can write a simple program to solve the problem: Using ML or DL for such a problem will be inaccurate or very inefficient
- ▶ Always start with simple models before moving to complex ones: Complex models have too many parameters. It is very difficult to know how they will behave in unexplored cases.
- ▶ Don't use DL if the problem is simple or if you get sufficient accuracy with a ML model: DL models are complex. Using DL for a simple problem is an overkill, the model will be inefficient
- ▶ Don't use DL if you have a small amount of data: You can't train a DL model with a small amount of data. The results will be really bad. Best situation is a lot of data with few very distinct parameters. This is not a magical method that solves all problems.

Representation Learning

The performance of simple machine learning algorithms depends heavily on the representation of the data they are given. For example, when logistic regression is used to recommend cesarean delivery, the AI system does not examine the patient directly. Instead, the doctor tells the system several pieces of relevant information, such as the presence or absence of a uterine scar. Each piece of information included in the representation of the patient is known as a feature.

Logistic regression learns how each of these features of the patient correlates with various outcomes. However, it cannot influence how features are defined in any way. If logistic regression were given an MRI scan of the patient, rather than the doctor's formalized report, it would not be able to make useful predictions. Individual pixels in an MRI scan have negligible correlation with any complications that might occur during delivery.

Representation Learning – Example

Suppose that we would like to write a program to detect cars in photographs. We know that cars have wheels, so we might like to use the presence of a wheel as a feature. Unfortunately, it is difficult to describe exactly what a wheel looks like in terms of pixel values. A wheel has a simple geometric shape, but its image may be complicated by shadows falling on the wheel, the sun glaring off the metal parts of the wheel, the fender of the car or an object in the foreground obscuring part of the wheel, and so on.

One solution to this problem is to use machine learning to discover not the mapping from representation to output but also the representation itself.

This approach is known as representation learning . Learned representations often result in much better performance than can be obtained with hand-designed representations.



Representation Learning – Example

They also enable AI systems to rapidly adapt to new tasks, with minimal human intervention. A representation learning algorithm can discover a good set of features for a simple task in minutes, or for a complex task in hours to months. Manually designing features for a complex task requires a great deal of human time and effort; it can take decades for an entire community of researchers.

Representation Learning

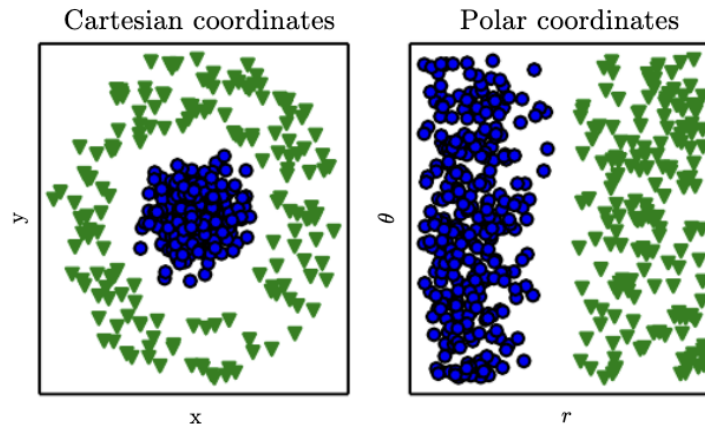


Figure 1.1: Example of different representations: suppose we want to separate two categories of data by drawing a line between them in a scatterplot. In the plot on the left, we represent some data using Cartesian coordinates, and the task is impossible. In the plot on the right, we represent the data with polar coordinates and the task becomes simple to solve with a vertical line. (Figure produced in collaboration with David Warde-Farley.)

<https://www.deeplearningbook.org/>

Representation Learning

A Venn diagram uses overlapping circles or other shapes to illustrate the logical relationships between two or more sets of items. Often, they serve to graphically organize things, highlighting how the items are similar and different.

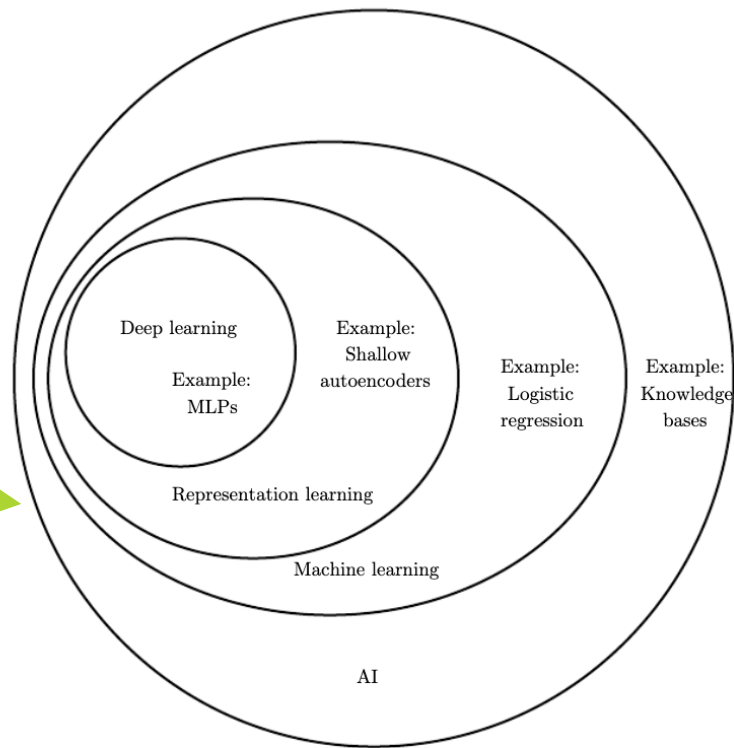


Figure 1.4: A Venn diagram showing how deep learning is a kind of representation learning, which is in turn a kind of machine learning, which is used for many but not all approaches to AI. Each section of the Venn diagram includes an example of an AI technology.

Definition

Deep Learning is a new area of Machine Learning research, which has been introduced with the objective of moving Machine Learning closer to one of its original goals

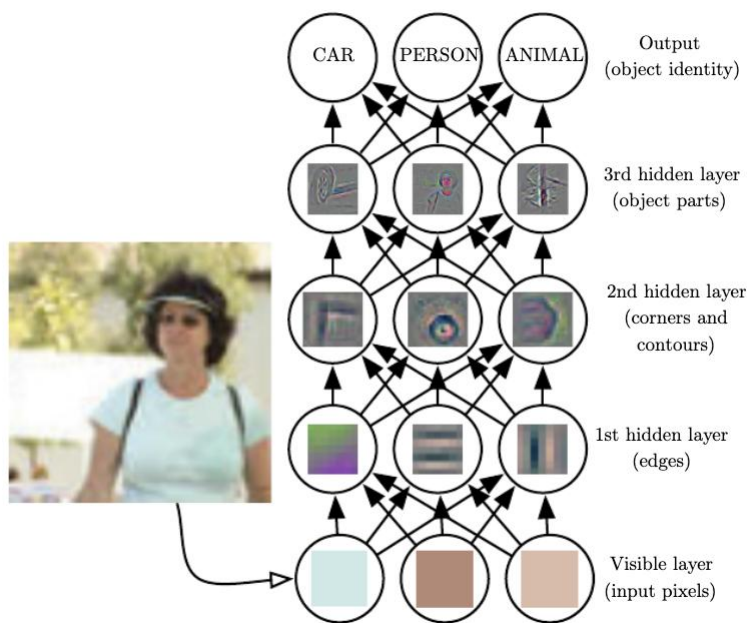
Deep Learning is about learning multiple levels of representation and abstraction that help to make sense of data such as images, sound, and text.

What is Deep Learning

Composition of non-linear transformation of the data.

Goal: Learn useful representations – features- directly from data

What is Deep Learning



<https://www.deeplearningbook.org/>

Given the pixels, the first layer can easily identify edges, by comparing the brightness of neighboring pixels.

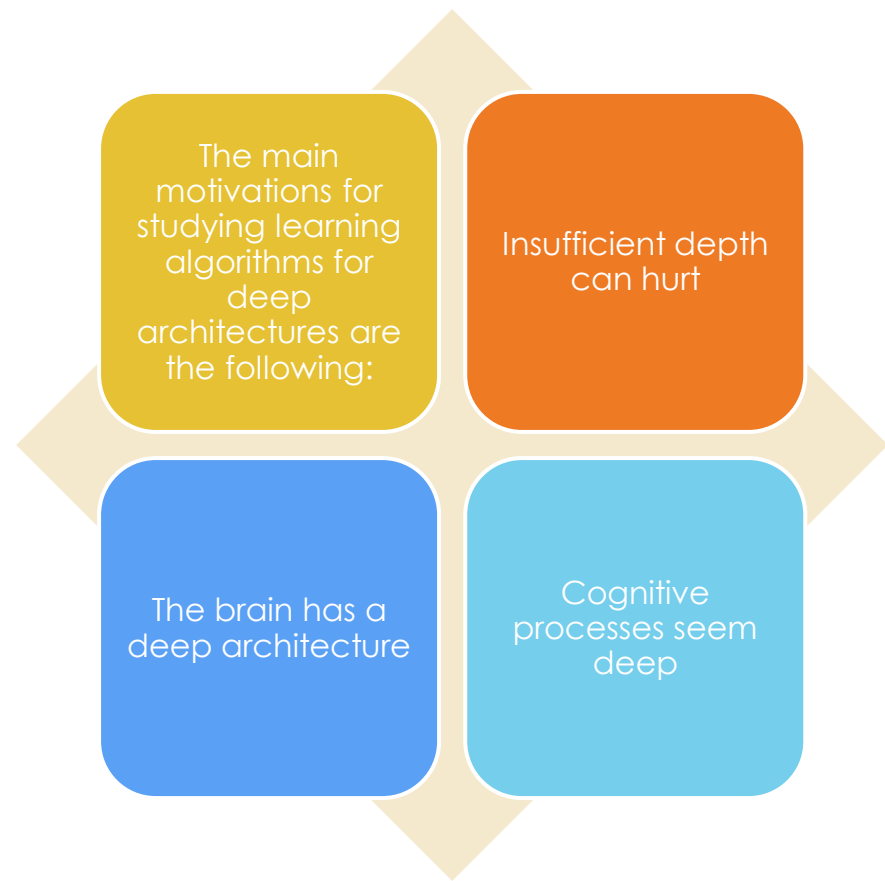
Given the first hidden layer's description of the edges, the second hidden layer can easily search for corners and extended contours, which are recognizable as collections of edges.

Given the second hidden layer's description of the image in terms of corners and contours, the third hidden layer can detect entire parts of specific objects, by finding specific collections of contours and corners.

Finally, this description of the image in terms of the object parts it contains can be used to recognize the objects present in the image.

Motivations

Motivations for Deep Architectures



Why Deep Learning ?

Why Deep Learning?

- Insufficient depth can hurt
 - ▶ With shallow architecture (SVM, NB, KNN, etc.), the required number of nodes in the graph (i.e. computations, and also number of parameters, when we try to learn the function) may grow very large.
 - ▶ Many functions that can be represented efficiently with a deep architecture cannot be represented efficiently with a shallow one.

Why Deep Learning ?

Why Deep Learning?

- The brain has a deep architecture
 - ▶ The visual cortex shows a sequence of areas each of which contains a representation of the input, and signals flow from one to the next.
 - ▶ Note that representations in the brain are in between dense distributed and purely local: they are **sparse**: about 1% of neurons are active simultaneously in the brain.

Why Deep Learning ?

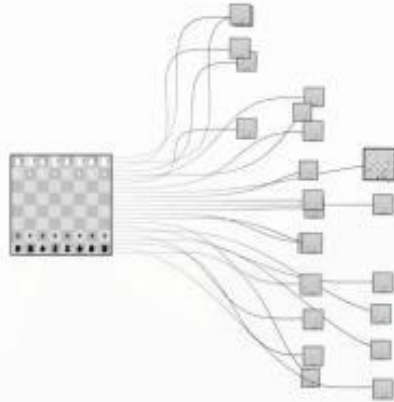
Why Deep Learning?

- Cognitive processes seem deep
 - ▶ Humans organize their ideas and concepts hierarchically.
 - ▶ Humans first learn simpler concepts and then compose them to represent more abstract ones.
 - ▶ Engineers break-up solutions into multiple levels of abstraction and processing



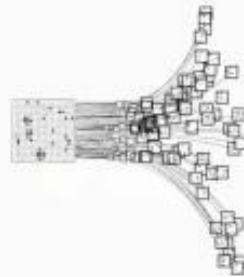
1. **Universal Learning Approach:** Because DL has the ability to perform in approximately all application domains, it is sometimes referred to as universal learning.
2. **Robustness:** In general, precisely designed features are not required in DL techniques. Instead, the optimized features are learned in an automated fashion related to the task under consideration. Thus, robustness to the usual changes of the input data is attained.
3. **Generalization:** Different data types or different applications can use the same DL technique, an approach frequently referred to as transfer learning (TL). Furthermore, it is a useful approach in problems where data is insufficient.
4. **Scalability:** DL is highly scalable. ResNet , which was invented by Microsoft, comprises of 1202 layers and is frequently applied at a supercomputing scale. Lawrence Livermore National Laboratory (LLNL), a large enterprise working on evolving frameworks for networks, adopted a similar approach, where thousands of nodes can be implemented.

Source: Sarker, I.H. Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions. SN COMPUT. SCI. 2, 420 (2021). <https://doi.org/10.1007/s42979-021-00815-1>



Chess: 10^{47}

Deep Blue, Feb 10, 1996



Go: 10^{170}

AlphaGo, March, 2016

Deep Blue

In 1996 it made history by defeating Russian grandmaster Garry Kasparov in one of their six games—the first time a computer had won a game against a world champion under tournament conditions.

In the 1997 rematch, it won the deciding sixth game in only 19 moves; its 3.5–2.5 victory (it won two games and had three draws) marked the first time a current world champion had lost a match to a computer under tournament conditions.

In its final configuration, the IBM RS6000/SP computer used 256 processors working in tandem, with an ability to evaluate 200 million chess positions per second.



Source: <https://www.britannica.com/topic/Deep-Blue>



AlphaGo

A computer program that combines advanced search tree with deep neural networks. These neural networks take a description of the Go board as an input and process it through a number of different network layers containing millions of neuron-like connections.

One neural network, the “policy network”, selects the next move to play. The other neural network, the “value network”, predicts the winner of the game. AlphaGo was introduced to numerous amateur games to help it develop an understanding of reasonable human play. Then it played against different versions of itself thousands of times, each time learning from its mistakes.

Over time, AlphaGo improved and became increasingly stronger and better at learning and decision-making. This process is known as reinforcement learning.

AlphaGo went on to defeat Go world champions in different global arenas and arguably became the greatest Go player of all time.*

* <https://www.deepmind.com/research/highlighted-research/alphago>



ChatGPT

Image source: <https://www.klippa.com/en/blog/information/what-is-chatgpt/>

Deepfake

Deepfake AI is a type of artificial intelligence used to create convincing image, audio and video hoaxes



Image source: <http://www.indianlink.com.au/wp-content/uploads/2015/01/hair2.jpg>

Deepfake

Deepfake AI is a type of artificial intelligence used to create convincing image, audio and video hoaxes



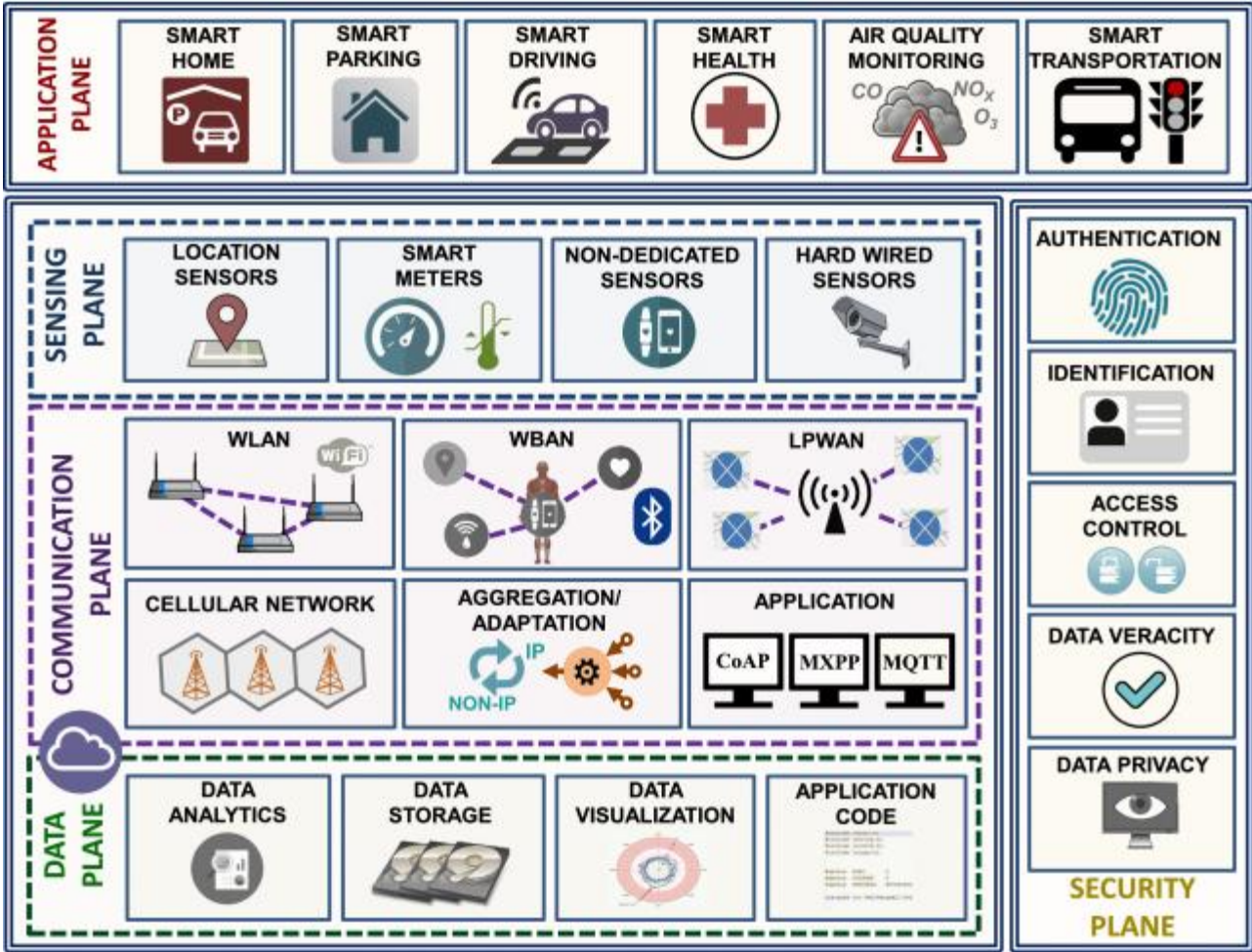
Image source: <https://www.nytimes.com/interactive/2020/11/21/science/artificial-intelligence-fake-people-faces.html>



When to apply deep learning

- Cases where human experts are not available.
- Cases where humans are unable to explain decisions made using their expertise (language understanding, medical decisions, and speech recognition).
- Cases where the problem solution updates over time (price prediction, stock preference, weather prediction, and tracking).
- Cases where solutions require adaptation based on specific cases (personalization, biometrics).
- Cases where size of the problem is extremely large and exceeds our inadequate reasoning abilities (sentiment analysis, matching ads to Facebook, calculation webpage ranks).

Smart cities



Source: Habibzadeh, H., Kaptan, C., Soyata, T., Kantarci, B., & Boukerche, A. (2019). Smart city system design: A comprehensive study of the application and data planes. *ACM Computing Surveys (CSUR)*, 52(2), 1-38.



Recommender Systems

One of the major families of applications of machine learning in the information technology sector is the ability to make recommendations of items to potential users or customers. Two major types of applications can be distinguished:

- online advertising
- item recommendations (often these recommendations are still for the purpose of selling a product).

Both rely on predicting the association between a user and an item, either to predict the probability of some action (the user buying the product, or some proxy for this action) or the expected gain (which may depend on the value of the product) if an ad is shown, or a recommendation is made regarding that product to that user.

Computer Vision

Computer vision is a field of artificial intelligence (AI) that enables computers and systems to derive meaningful information from digital images.

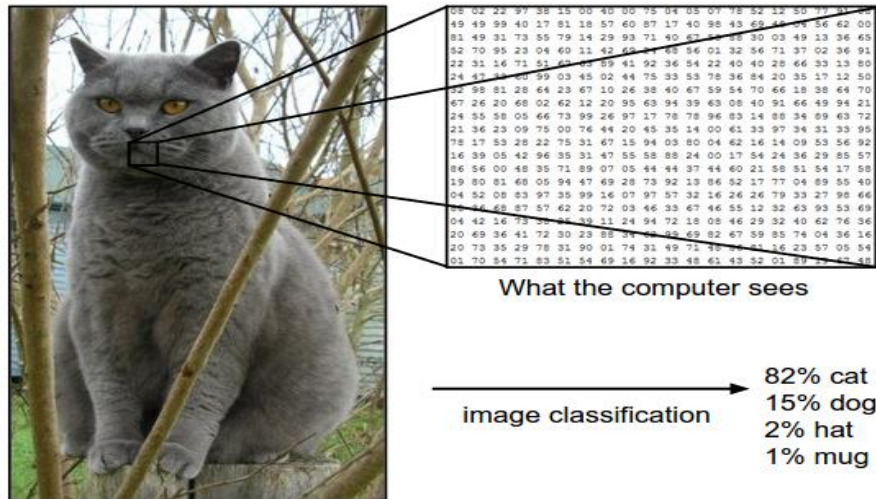


Image source: <https://cs231n.github.io/classification/>

Convolutional Neural Networks perform exceptionally well for computer vision tasks



Autonomous vehicles

AI-powered robotaxis are expected to create a \$2 trillion market worldwide by 2030, according to global financial services firm UBS.

Self-driving cars use camera-based machine vision systems and radar- and lidar-based detection units to perceive, understand, and safely navigate their surrounding environment.

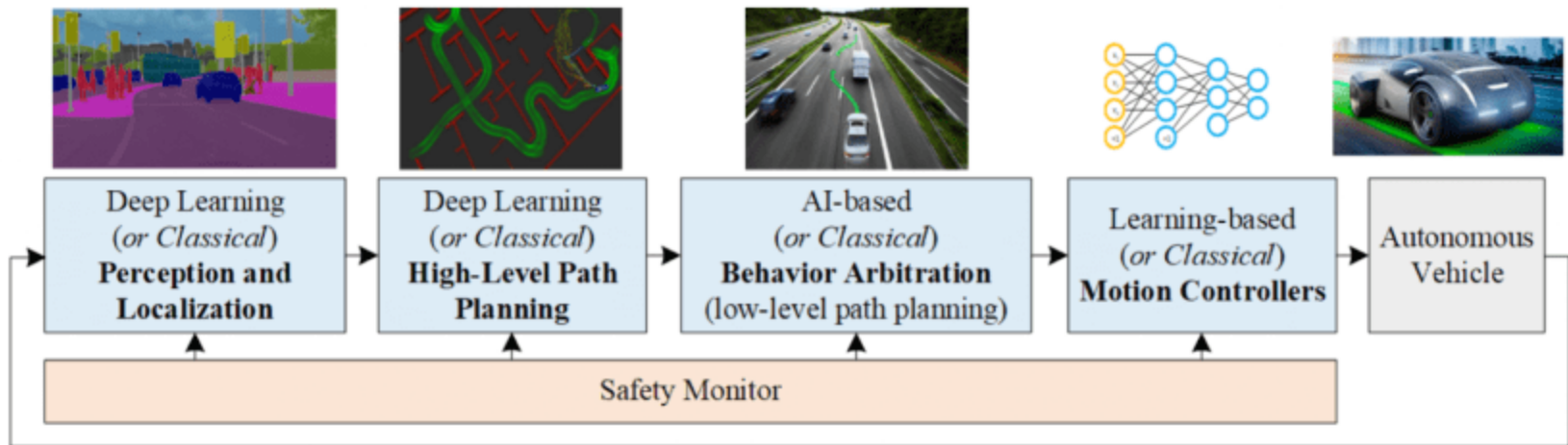
The goal is to optimize performance of self-driving car perception applications such as lane navigation and pedestrian detection.

A semantic segmentation neural network has to be build and trained, to identify objects such as roads, pedestrians, and other vehicles.

There are many platforms to deploy the implemented neural network. For example the [NVIDIA DRIVE AGX™](#).

AI applications for autonomous vehicles concern a variety of autonomous driving scenarios such as highway driving, city roads, and parking.

Autonomous vehicles



Source: Grigorescu, S., Trasnea, B., Cocias, T., & Macesanu, G. (2020). A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37(3), 362-386.

Autonomous vehicles



Image source: <https://spectrum.ieee.org/deep-learning-makes-driverless-cars-better-at-spotting-pedestrians>

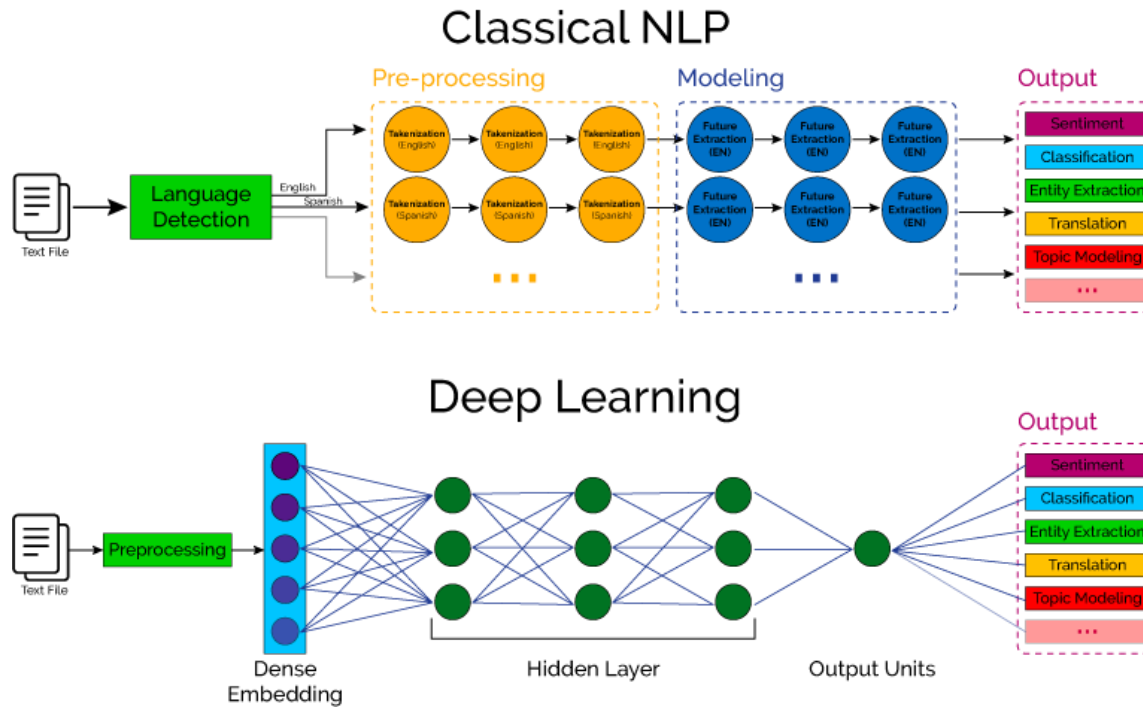
Autonomous vehicles



Image source: <https://techxplore.com/news/2020-05-deep-image-recognition-ability-self-driving.html>

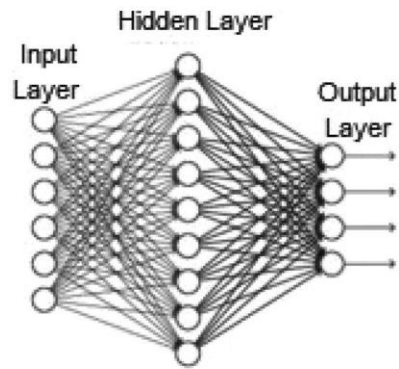
NLP

Natural language processing (NLP) is the use of human languages, such as English or French, by a computer. It is the ability of a computer program to understand human language as it is spoken and written -- referred to as natural language.

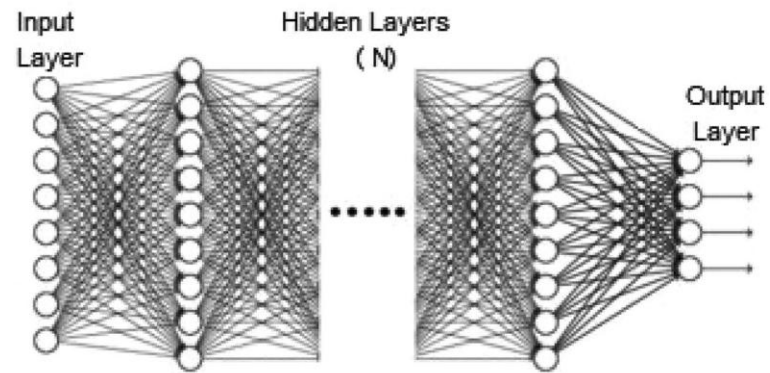


Source: <https://www.nsuchaud.fr/2018/03/difference-between-classical-nlp-deep-learning-nlp/>

Shallow vs Deep network



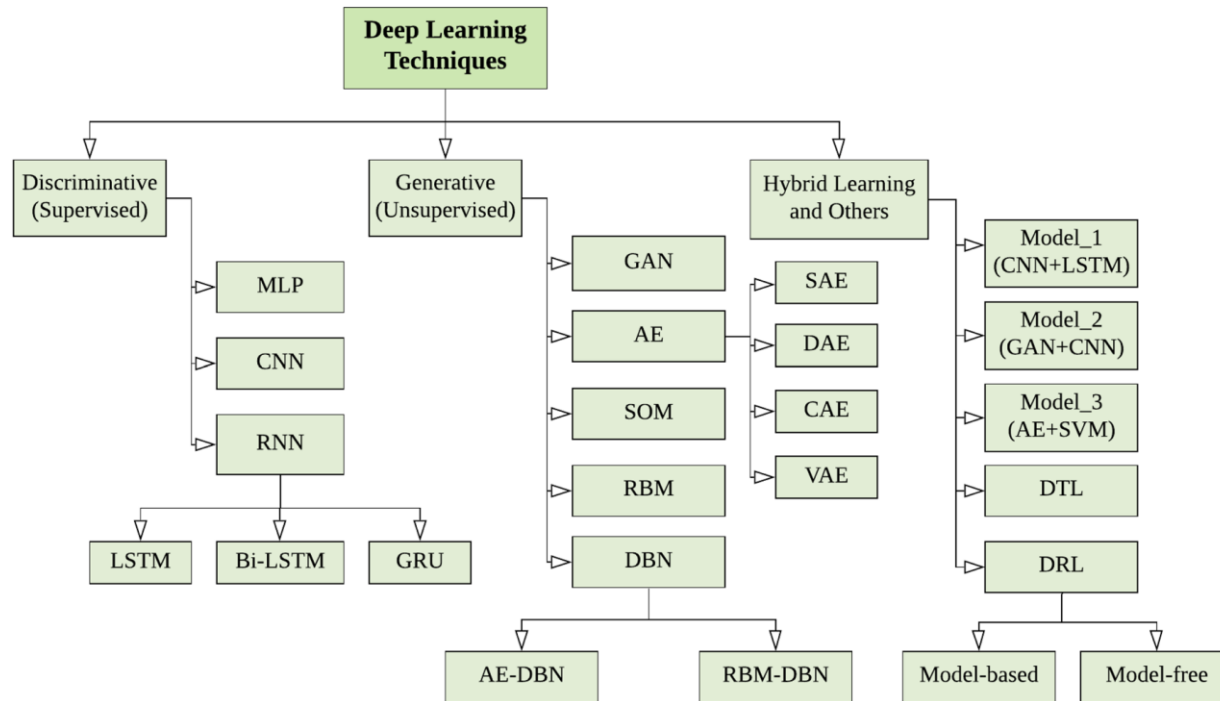
(a) A Shallow Network



(b) A Deep Neural Network

Source: Sarker, I.H. Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions. SN COMPUT. SCI. 2, 420 (2021). <https://doi.org/10.1007/s42979-021-00815-1>

Taxonomy of deep learning techniques



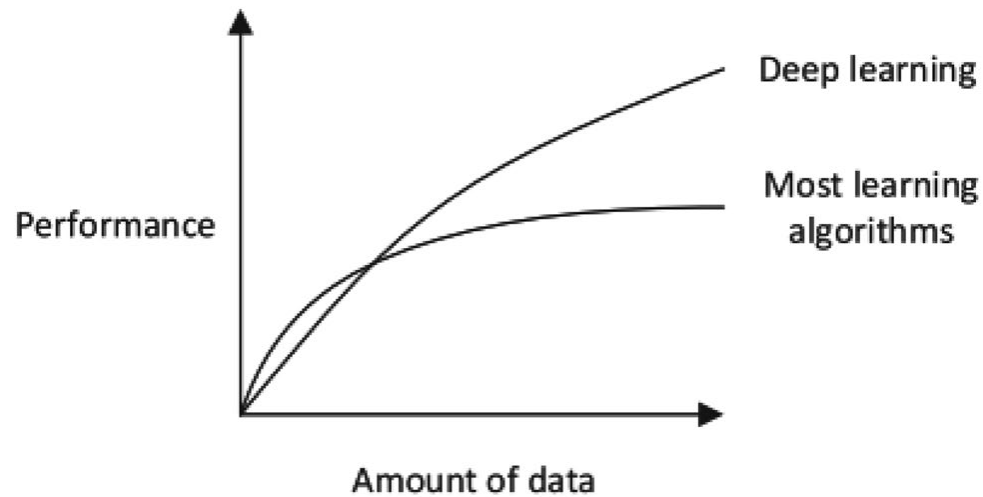
Source: Sarker, I.H. Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions. SN COMPUT. SCI. 2, 420 (2021). <https://doi.org/10.1007/s42979-021-00815-1>

Table 1 A summary of deep learning tasks and methods in several popular real-world applications areas

Application areas	Tasks	Methods	References
Healthcare and Medical applications	Regular health factors analysis	CNN-based	Ismail et al. [48]
	Identifying malicious behaviors	RNN-based	Xue et al. [129]
	Coronary heart disease risk prediction	Autoencoder based	Amarbayasgalan et al. [6]
	Cancer classification	Transfer learning based	Sevakula et al. [110]
Natural Language Processing	Diagnosis of COVID-19	CNN and BiLSTM based	Aslan et al. [10]
	Detection of COVID-19	CNN-LSTM based	Islam et al. [47]
	Text summarization	Auto-encoder based	Yousefi et al. [130]
	Sentiment analysis	CNN-LSTM based	Wang et al. [120]
Speech recognition	Sentiment analysis	CNN and Bi-LSTM based	Minaee et al. [78]
	Aspect-level sentiment classification	Attention-based LSTM	Wang et al. [124]
	Distant speech recognition	Attention-based LSTM	Zhang et al. [135]
	Speech emotion classification	Transfer learning based	Latif et al. [63]
Cybersecurity	Emotion recognition from speech	CNN and LSTM based	Satt et al. [109]
	Zero-day malware detection	Autoencoders and GAN based	Kim et al. [54]
	Security incidents and fraud analysis	SOM-based	Lopez et al. [70]
	Android malware detection	Autoencoder and CNN based	Wang et al. [122]
IoT and Smart cities	intrusion detection classification	DBN-based	Wei et al. [125]
	DoS attack detection	RBM-based	Imamverdiyev et al. [46]
	Suspicious flow detection	Hybrid deep-learning-based	Garg et al. [29]
	Network intrusion detection	AE and SVM based	Al et al. [4]
Smart Agriculture	Smart energy management	CNN and Attention mechanism	Abdel et al. [2]
	Particulate matter forecasting	CNN-LSTM based	Huang et al. [43]
	Smart parking system	CNN-LSTM based	Piccialli et al. [85]
	Disaster management	DNN-based	Aqib et al. [8]
Business and Financial Services	Air quality prediction	LSTM-RNN based	Kok et al. [61]
	Cybersecurity in smart cities	RBM, DBN, RNN, CNN, GAN	Chen et al. [15]
	A smart agriculture IoT system	RL-based	Bu et al. [11]
	Plant disease detection	CNN-based	Ale et al. [5]
Virtual Assistant and Chatbot Services	Automated soil quality evaluation	DNN-based	Sumathi et al. [115]
	Predicting customers' purchase behavior	DNN based	Chaudhuri [14]
	Stock trend prediction	CNN and LSTM based	anuradha et al. [7]
	Financial loan default prediction	CNN-based	Deng et al. [23]
Object Detection and Recognition	Power consumption forecasting	LSTM-based	Shao et al. [112]
	An intelligent chatbot	Bi-RNN and Attention model	Dhyani et al. [24]
	Virtual listener agent	GRU and LSTM based	Huang et al. [44]
	Smart blind assistant	CNN-based	Rahman et al. [88]
Recommendation and Intelligent system	Object detection in X-ray images	CNN-based	Gu et al. [35]
	Object detection for disaster response	CNN-based	Pi et al. [84]
	Medicine recognition system	CNN-based	Chang et al. [12]
	Face recognition in IoT-cloud environment	CNN-based	Masud et al. [76]
Intelligent transportation system	Food recognition system	CNN-based	Liu et al. [68]
	Affect recognition system	DBN-based	Kawde et al. [53]
	Facial expression analysis	CNN and LSTM based	Li et al. [67]
	Hybrid recommender system	DNN-based	Kiran et al. [57]
Intelligent transportation system	Visual recommendation and search	CNN-based	Shankar et al. [111]
	Recommendation system	CNN and Bi-LSTM based	Rosa et al. [91]
	Intelligent system for impaired patients	RL-based	Naeem et al. [79]
	Intelligent transportation system	CNN-based	Wang et al. [123]

Source: Sarker, I.H. Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions. SN COMPUT. SCI. 2, 420 (2021). <https://doi.org/10.1007/s42979-021-00815-1>

Performance comparison between deep learning and other machine learning algorithms



Source: Sarker, I.H. Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions. SN COMPUT. SCI. 2, 420 (2021). <https://doi.org/10.1007/s42979-021-00815-1>



Problems with data

- Insufficient data
- Too much data
- Non- Representative data
- Missing data
- Duplicate data
- Outliers

Models trained with insufficient data perform poorly while making predictions. It may lead to 2 cases:

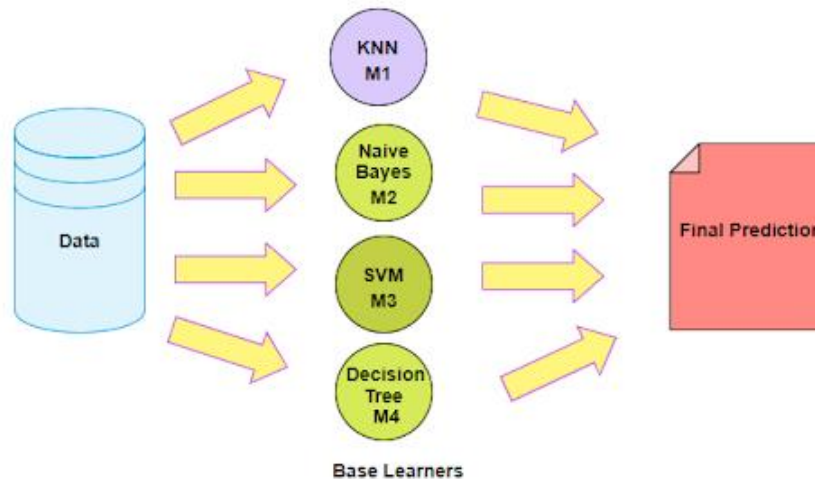
Over-fitting: Here the training model reads the data too much for too little data. This means the training model actually memorizes the patterns. It has low training errors and high test errors. Does not work well in the real world.

Under-fitting: It builds a very simple model. This means the data are not trained to our expectations. The model is unable to capture relationship in data and has no predictive power.

Problems with data - Insufficient data

Model Complexity:

- Build a simple model with fewer parameters. This method is less susceptible to over-fitting. Example: Naive Bayes, Linear Regression.
- Use an Ensemble learning technique: It is defined as several learners are combined to obtain a better performance than any individual learners. It is often used to improve classification and prediction.



Source: Google Images – Ensemble Learning



Problems with data - Insufficient data

Transfer Learning:

Transfer Learning uses a pre-built model, which is then tweaked on the small dataset that you have.

It is also defined as the practice of **reusing a trained Neural Network**, that solves a similar problem to yours, usually leaving the network architecture unchanged and reusing some of the model weights.

It is very much **useful when the new dataset is small** and not sufficient to train the model from scratch.



Problems with data - Insufficient data

Data Augmentation:

Data Augmentation takes the pre-existing samples and changes them in some way to create new samples and also increase the number of training samples and typically used with Image data.

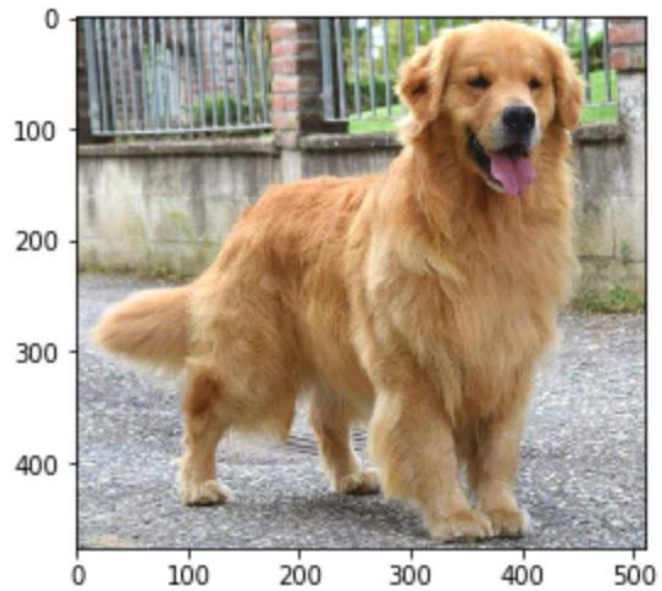
These image processing options are often used as pre-processing techniques to make image classification models built using CNN are robust.

Problems with data - Insufficient data

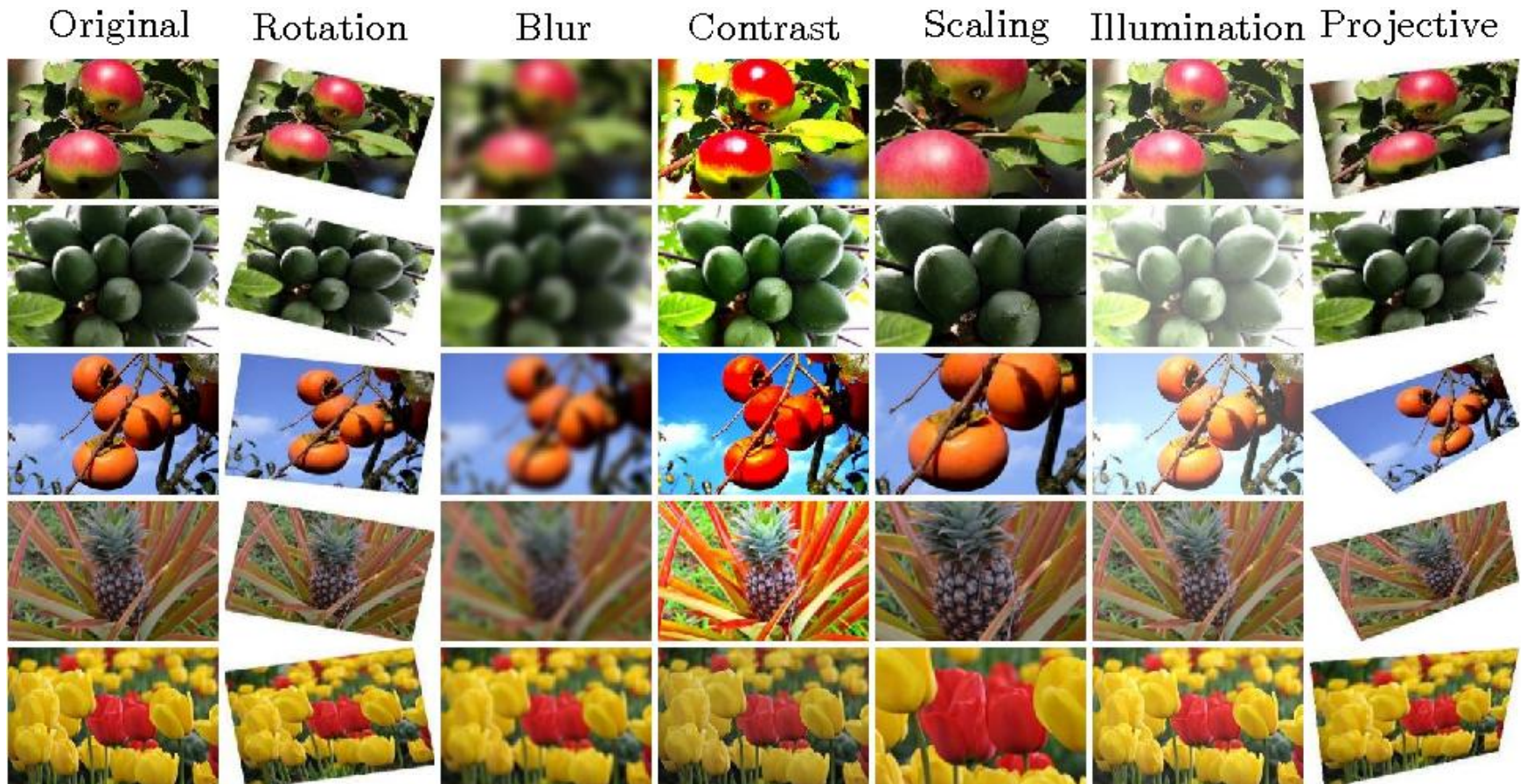
Classic image processing activities for data augmentation are:

- padding
- random rotating
- re-scaling,
- vertical and horizontal flipping
- translation (image is moved along X, Y direction)
- cropping
- zooming
- darkening & brightening/color modification
- grayscaling
- changing contrast
- adding noise
- random erasing
- affine Transforms

Problems with data - Insufficient data



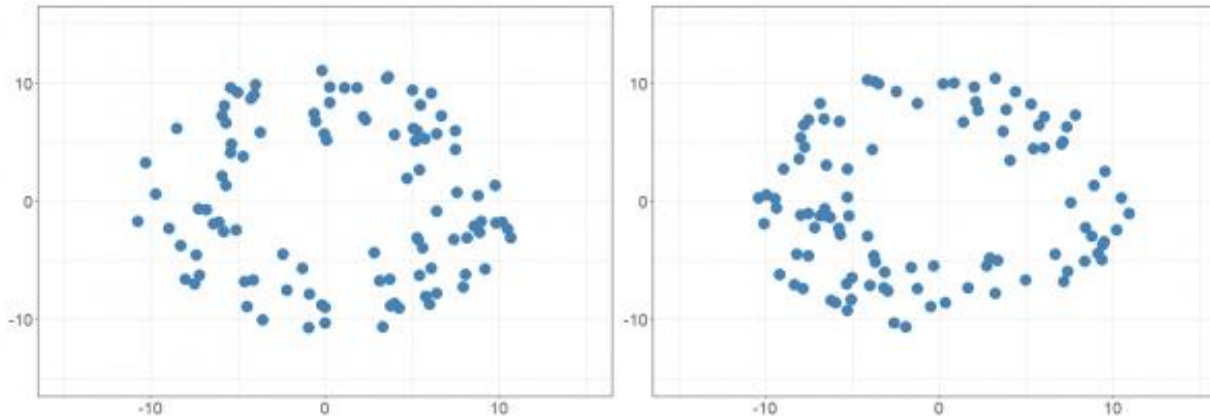
Problems with data - Insufficient data



Problems with data - Insufficient data

Synthetic Data:

Synthetic data generally refers to artificially generating samples which mimic the real-world data. This may induce bias in existing data.



Original data

Synthetic data

The synthetic data retains the structure of the original data but is not the same

Source: Google Images – Synthetic data

Problems with data - Insufficient data

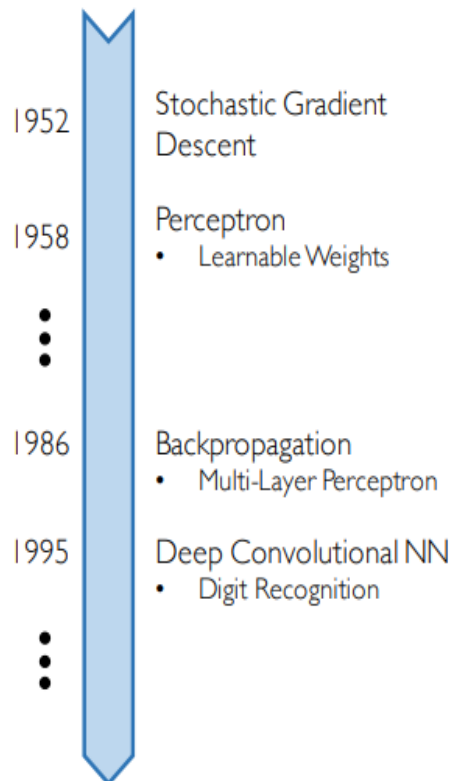
Original Data



Synthetic Data



Why Now ?



Neural Networks date back decades, so why the resurgence?

1. Big Data

- Larger Datasets
- Easier Collection & Storage

IMAGENET



WIKIPEDIA
The Free Encyclopedia



2. Hardware

- Graphics Processing Units (GPUs)
- Massively Parallelizable

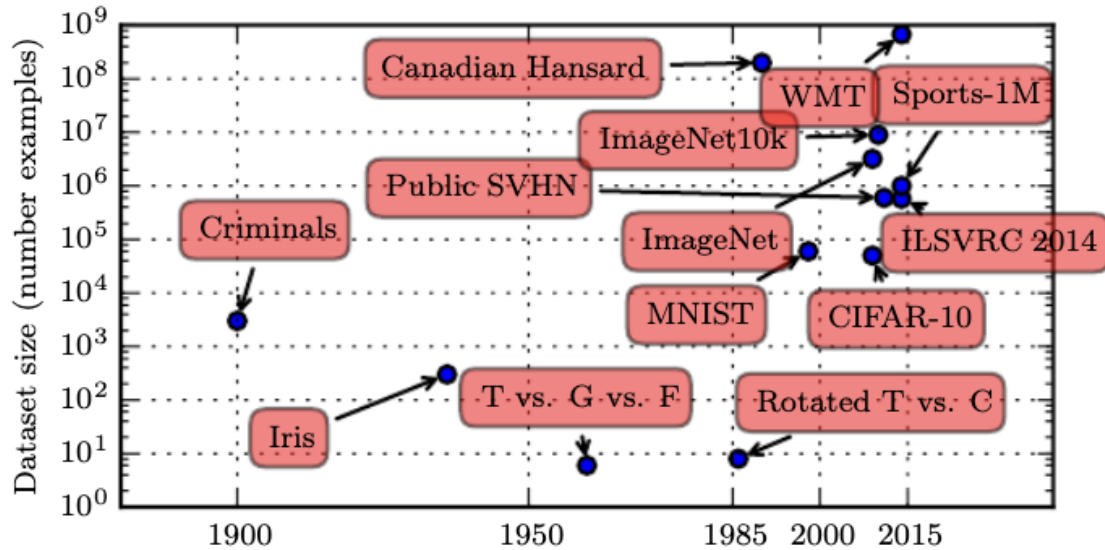


3. Software

- Improved Techniques
- New Models
- Toolboxes

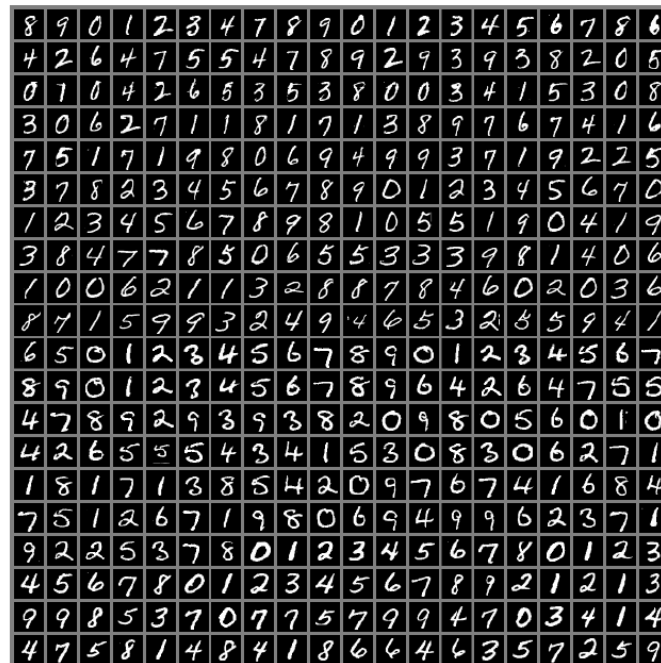


Data



Increasing dataset size over time – Source: <https://www.deeplearningbook.org/>

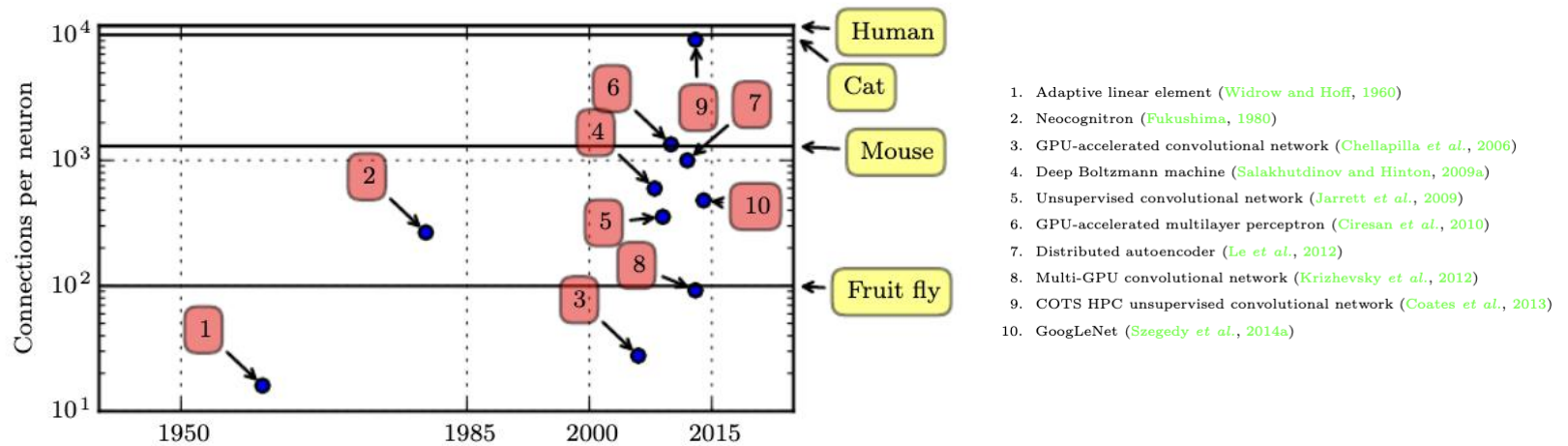
Data – Example



8	9	0	1	2	3	4	7	8	9	0	1	2	3	4	5	6	7	8	6
4	2	6	4	7	5	5	4	7	8	9	2	9	3	9	3	8	2	0	5
0	7	0	4	2	6	5	3	5	3	8	0	0	3	4	1	5	3	0	8
3	0	6	2	7	1	1	8	1	7	1	3	8	9	7	6	7	4	1	6
7	5	1	7	1	9	8	0	6	9	4	9	9	3	7	1	9	2	2	5
3	7	8	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	0
1	2	3	4	5	6	7	8	9	8	1	0	5	5	1	9	0	4	1	9
3	8	4	7	7	8	5	0	6	5	5	3	3	3	9	8	1	4	0	6
1	0	0	6	2	1	1	3	2	8	8	7	8	4	6	0	2	0	3	6
8	7	1	5	9	9	3	2	4	9	4	6	5	3	2	8	5	9	4	1
6	5	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7
8	9	0	1	2	3	4	5	6	7	8	9	6	4	2	6	4	7	5	5
4	7	8	9	2	9	3	9	3	8	2	0	9	8	0	5	6	0	1	0
4	2	6	5	5	5	4	3	4	1	5	3	0	8	3	0	6	2	7	1
1	8	1	7	1	3	8	5	4	2	0	9	7	6	7	4	1	6	8	4
7	5	1	2	6	7	1	9	8	0	6	9	4	9	9	6	2	3	7	1
9	2	2	5	3	7	8	0	1	2	3	4	5	6	7	8	0	1	2	3
4	5	6	7	8	0	1	2	3	4	5	6	7	8	9	2	1	2	1	3
9	9	8	5	3	7	0	7	7	5	7	9	9	4	7	0	3	4	1	4
4	7	5	8	1	4	8	4	1	8	6	6	4	6	3	5	7	2	5	9

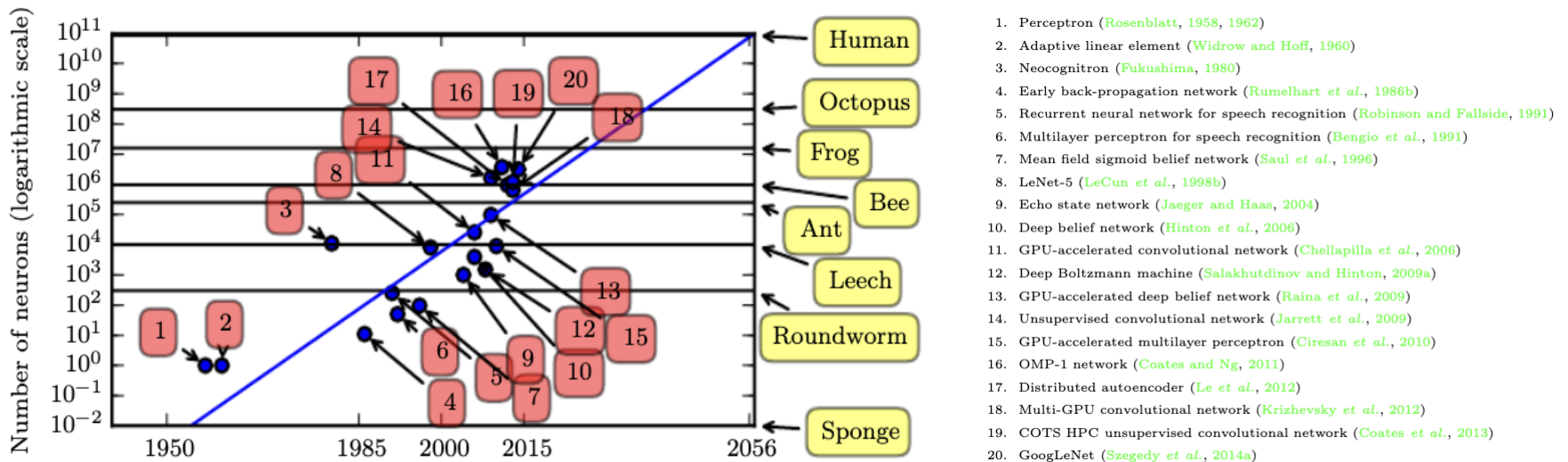
Sample images from MNIST dataset

Neuron connections



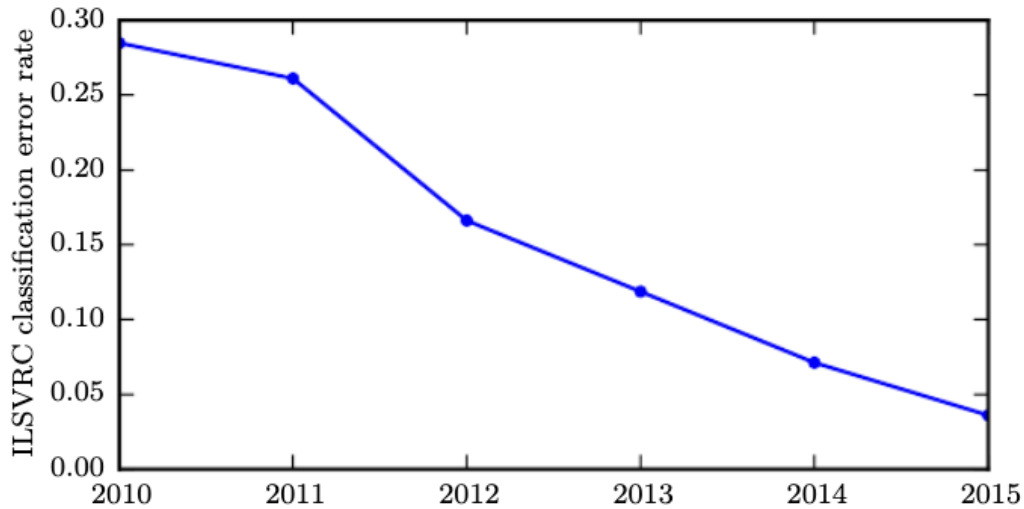
Number of connection per neuron over time - Source: <https://www.deeplearningbook.org/>

Network Size



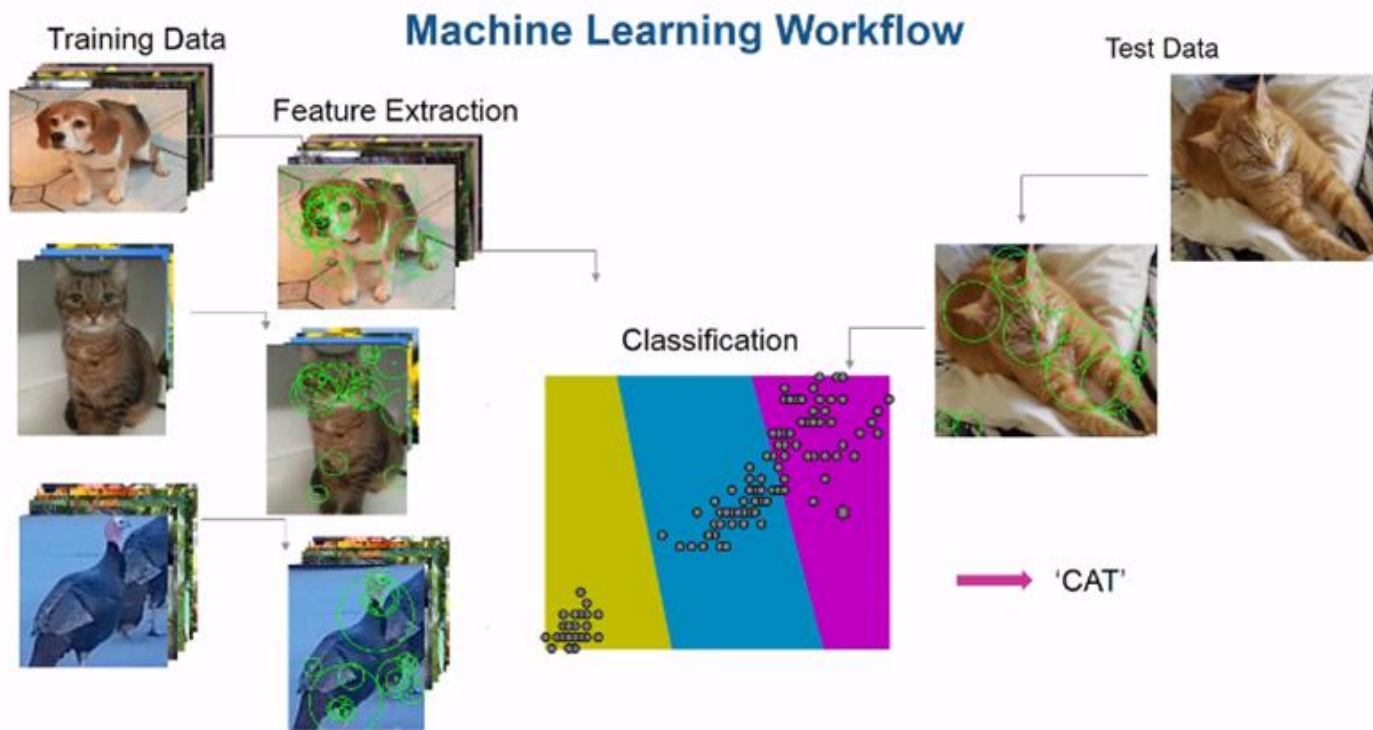
Increasing neural network size over time. - Source: <https://www.deeplearningbook.org/>

Network error



Decreasing error over time. - Source: <https://www.deeplearningbook.org/>


Until Now...



Recall : What is Deep Learning

Composition of non-linear transformation of the data.

Goal: Learn useful representations – features- directly from data

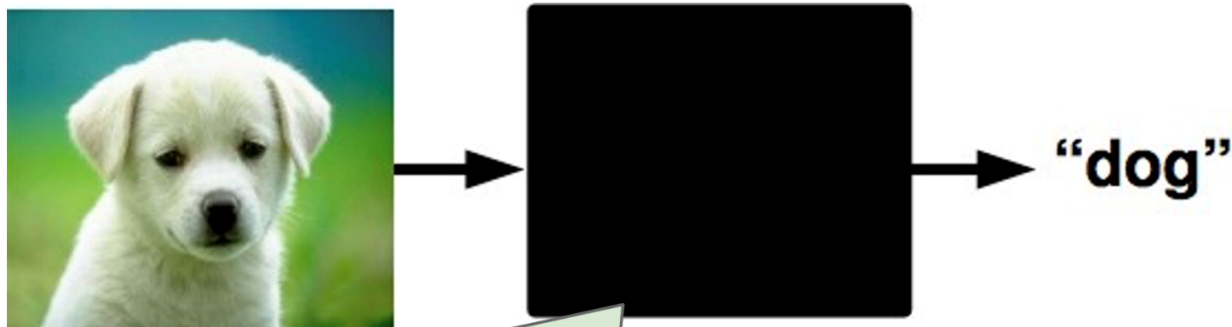


Deep learning is about learning
feature representation in a
compositional manner.

But wait,
why learn features?

The Black Box in a Traditional Recognition Approach

The Black Box in a Traditional Recognition Approach



Preprocessing

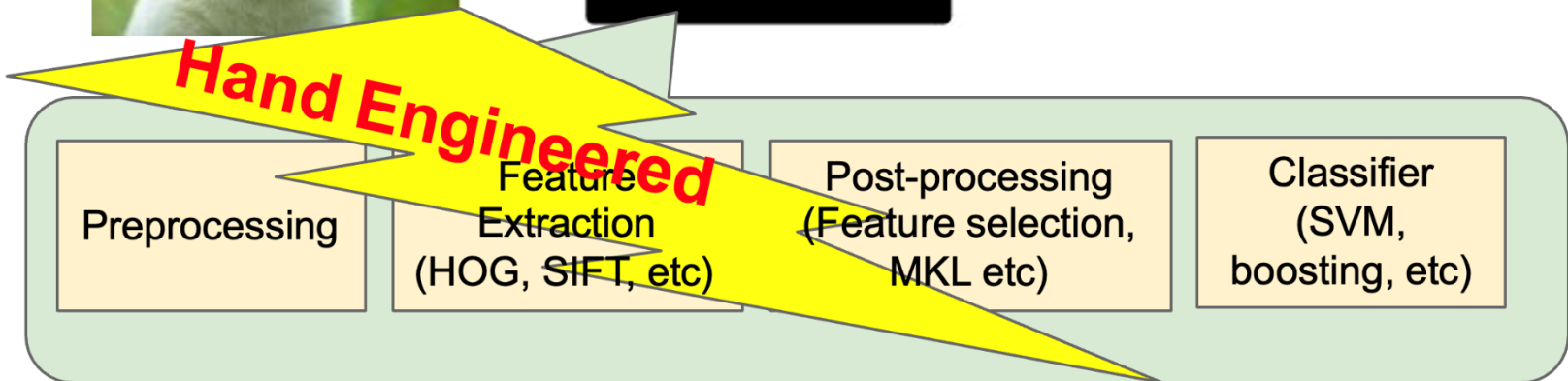
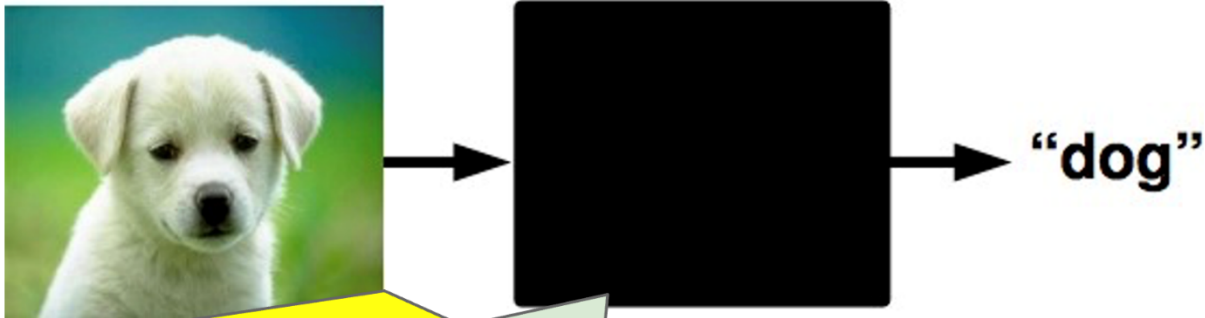
Feature
Extraction
(HOG, SIFT, etc)

Post-processing
(Feature selection,
MKL etc)

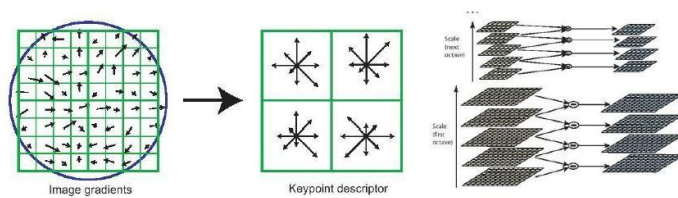
Classifier
(SVM,
boosting, etc)

The Black Box in a Traditional Recognition Approach

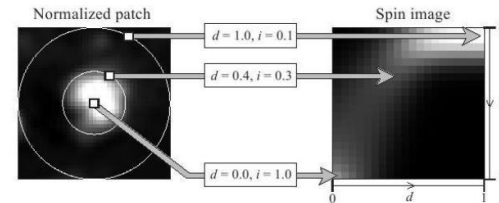
The Black Box in a Traditional Recognition Approach



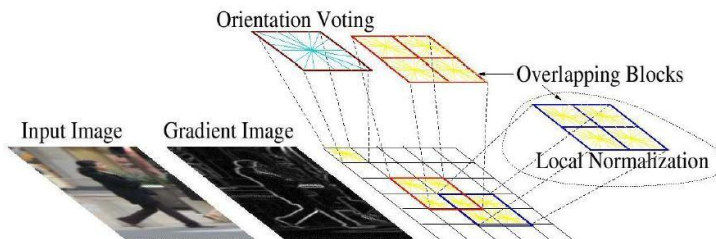
Computer vision features



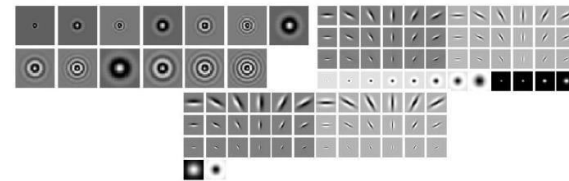
SIFT



Spin image



HoG



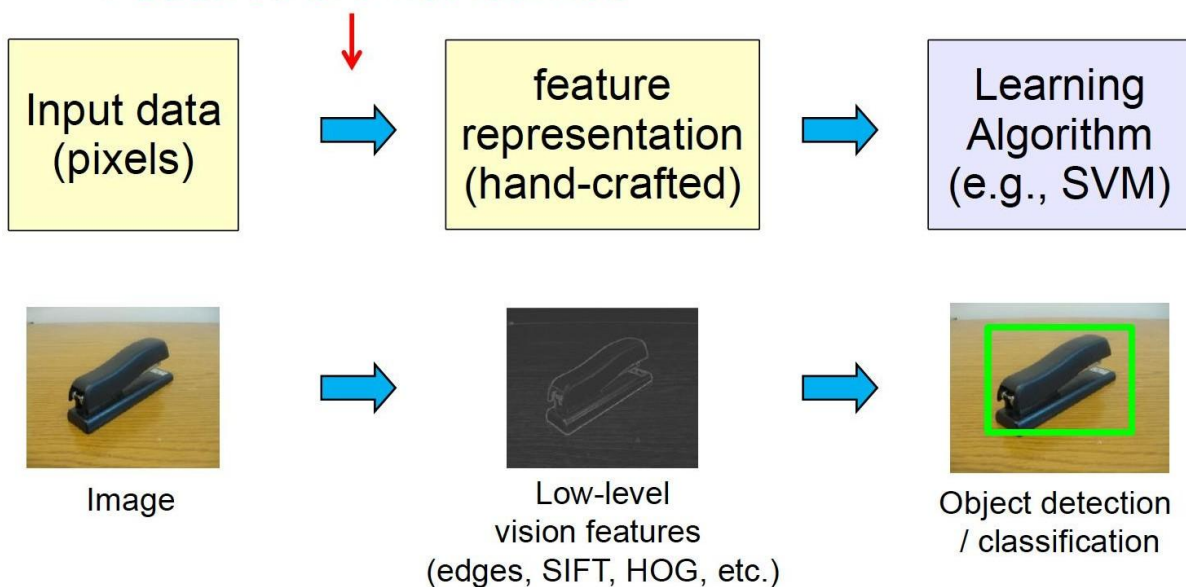
Textons

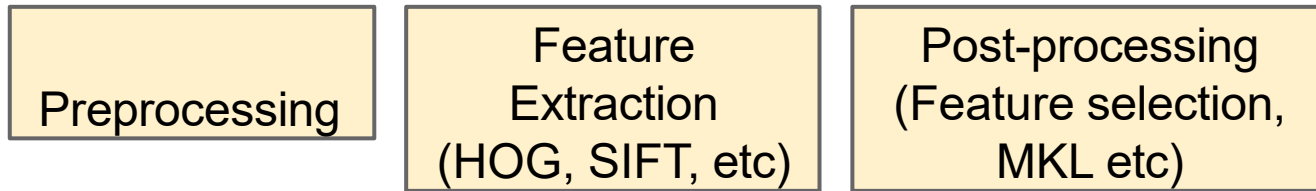
and many others:

SURF, MSER, LBP, Color-SIFT, Color histogram, GLOH,

Traditional Recognition Approach

Features are not learned





- Most critical for accuracy
 - Most time-consuming in development
 - What is the best feature???
 - What is next?? Keep on crafting better features?
- ⇒ Let's learn feature representation directly from data.

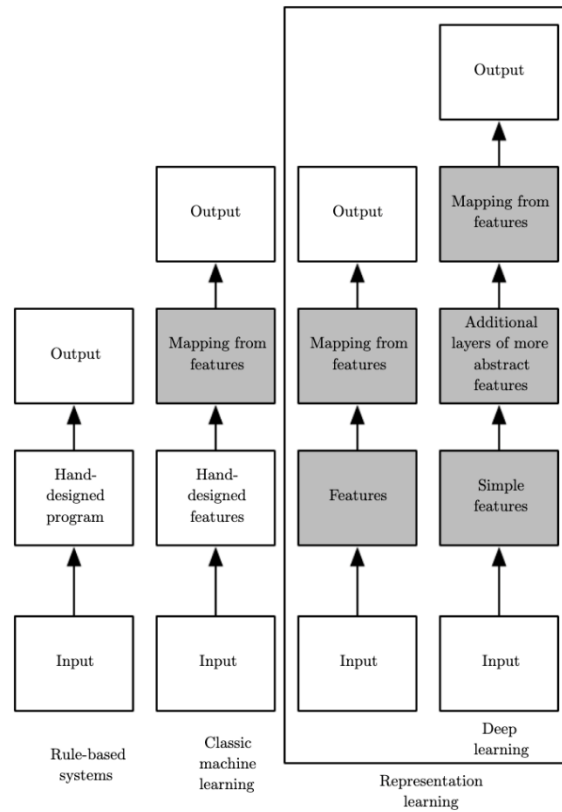


Figure 1.5: Flowcharts showing how the different parts of an AI system relate to each other within different AI disciplines. Shaded boxes indicate components that are able to learn from data.

<https://www.deeplearningbook.org/>

Labradoodle or fried chicken



Puppy or bagel



Sheepdog or mop



Chihuahua or muffin



Barn owl or apple



Parrot or guacamole



Raw chicken or Donald Trump



Feature Engineering vs. Learning

- Feature engineering is the process of using domain knowledge of the data to create features that make machine learning algorithms work.
- “Coming up with features is difficult, time-consuming, requires expert knowledge.”

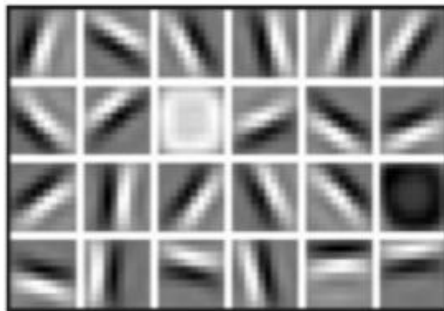
--Andrew Ng

Deep Learning = Learning Hierarchical Representations

Hand engineered features are time consuming, brittle and not scalable in practice

Can we learn the **underlying features** directly from data?

Low Level Features



Lines & Edges

Mid Level Features



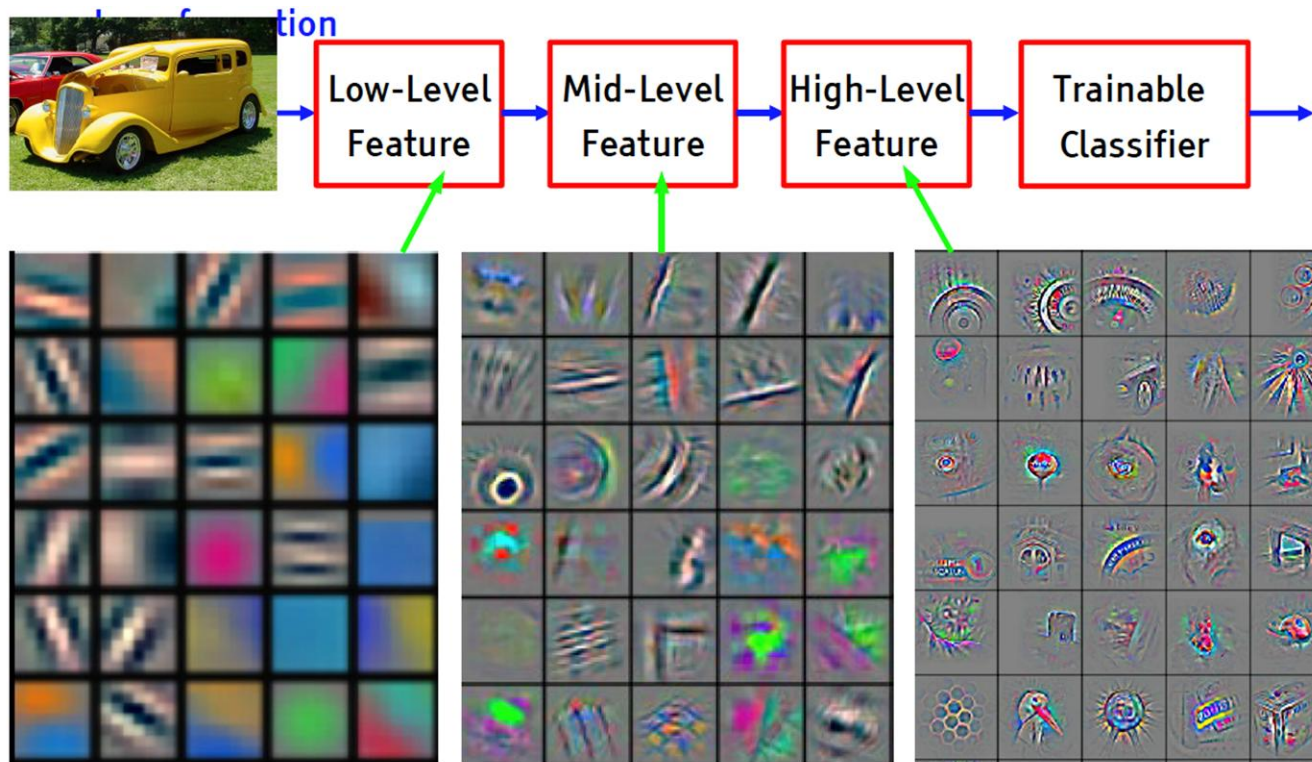
Eyes & Nose & Ears

High Level Features



Facial Structure

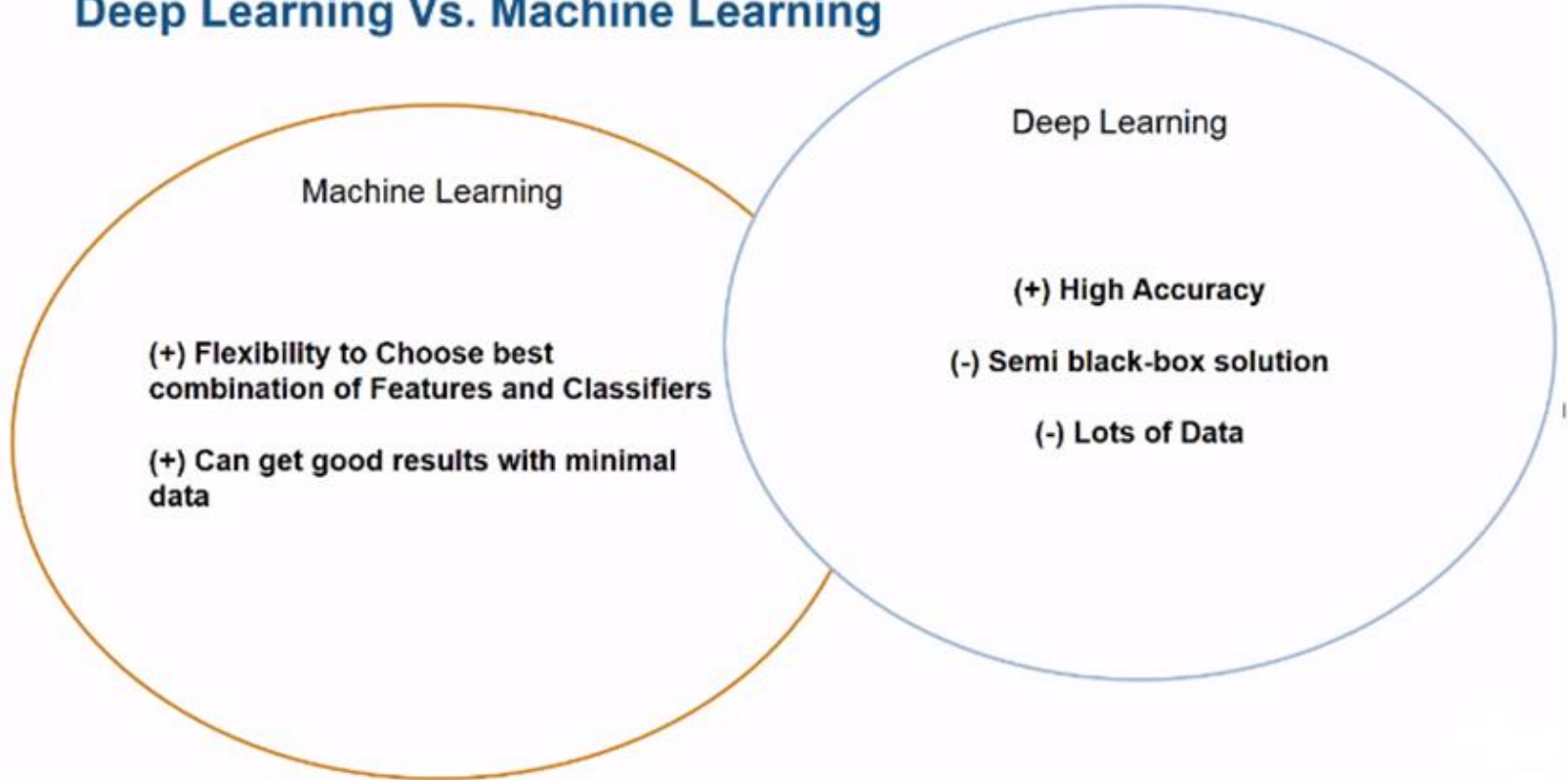
Deep Learning = Learning Hierarchical Representations



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Deep Learning vs Machine Learning

Deep Learning Vs. Machine Learning



Deep Learning AND Machine Learning

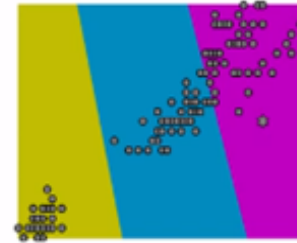
Deep Learning + Machine Learning Combined



Feature Learning

Deep Learning

Classification



Machine Learning

Deep Learning AND Machine Learning

Deep Learning AND Machine Learning

**Combination
Approach:**

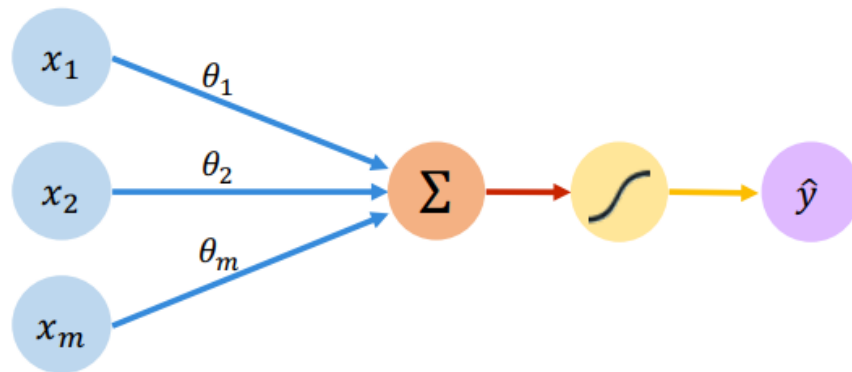
**Deep Learning as a
Feature Extractor**

**Machine Learning
to create classifier**

Remember: your results may vary
Do what's best for your problem and data!

Perceptron

The Perceptron: Forward Propagation



Linear combination of inputs

$$\hat{y} = g \left(\sum_{i=1}^m x_i \theta_i \right)$$

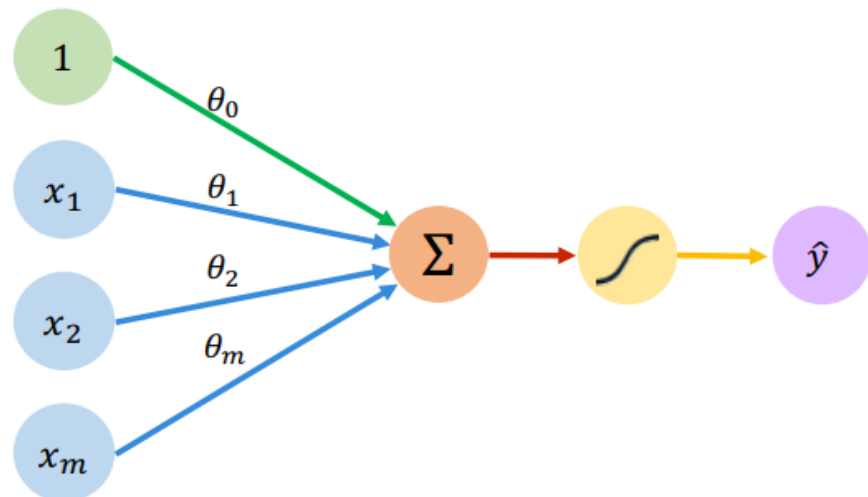
Output

Non-linear activation function

Inputs Weights Sum Non-Linearity Output

Perceptron

The Perceptron: Forward Propagation



Inputs Weights Sum Non-Linearity Output

$$\hat{y} = g \left(\theta_0 + \sum_{i=1}^m x_i \theta_i \right)$$

Output

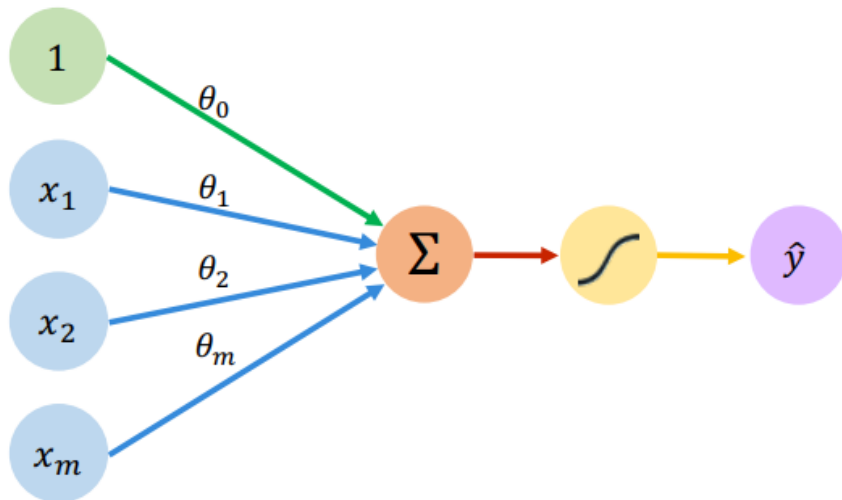
Linear combination of inputs

Non-linear activation function

Bias

Perceptron

The Perceptron: Forward Propagation



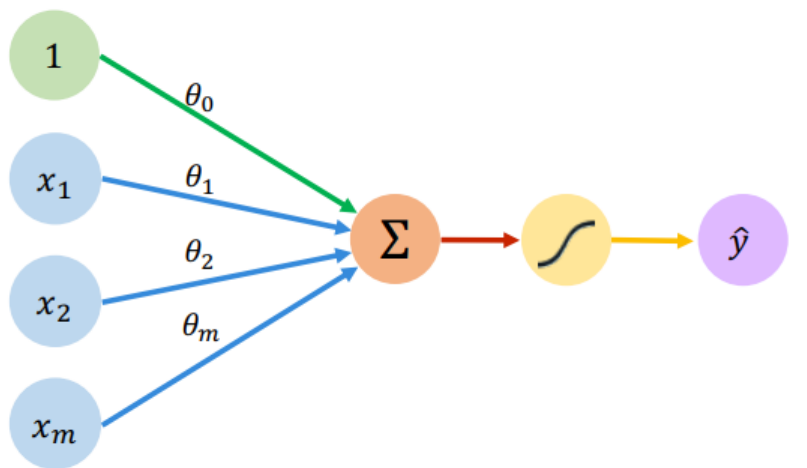
$$\hat{y} = g \left(\theta_0 + \sum_{i=1}^m x_i \theta_i \right)$$

$$\hat{y} = g \left(\theta_0 + \mathbf{X}^T \boldsymbol{\theta} \right)$$

$$\text{where: } \mathbf{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \text{ and } \boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_m \end{bmatrix}$$

Inputs Weights Sum Non-Linearity Output

The Perceptron: Forward Propagation



Inputs Weights Sum Non-Linearity Output

Activation Functions

$$\hat{y} = g(\theta_0 + \mathbf{X}^T \boldsymbol{\theta})$$

- Example: sigmoid function

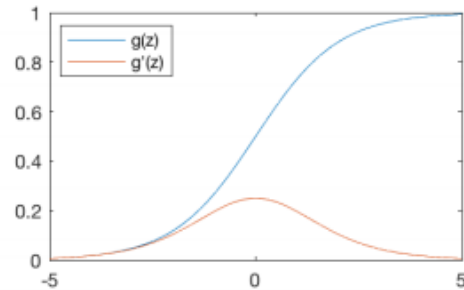
$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$



Activation Functions

Common Activation Functions

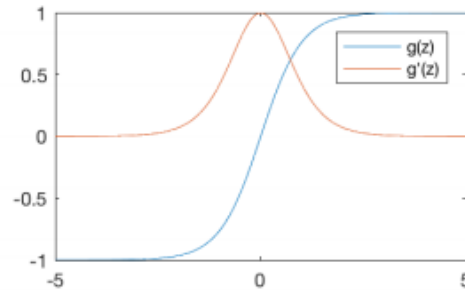
Sigmoid Function



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

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

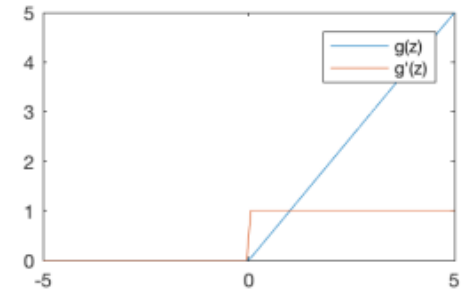
Hyperbolic Tangent



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

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

Rectified Linear Unit (ReLU)



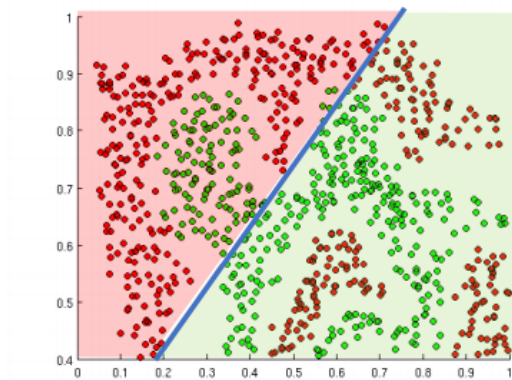
$$g(z) = \max(0, z)$$

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

Activation Functions

Importance of Activation Functions

The purpose of activation functions is to *introduce non-linearities* into the network

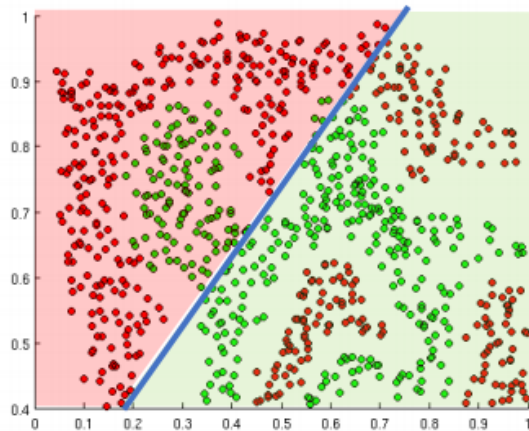


Linear Activation functions produce linear decisions no matter the network size

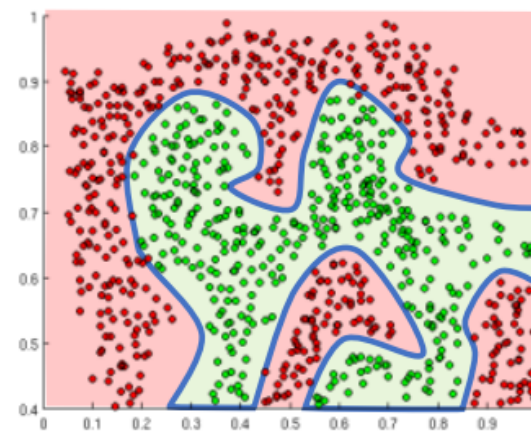
Activation Functions

Importance of Activation Functions

*The purpose of activation functions is to **introduce non-linearities** into the network*



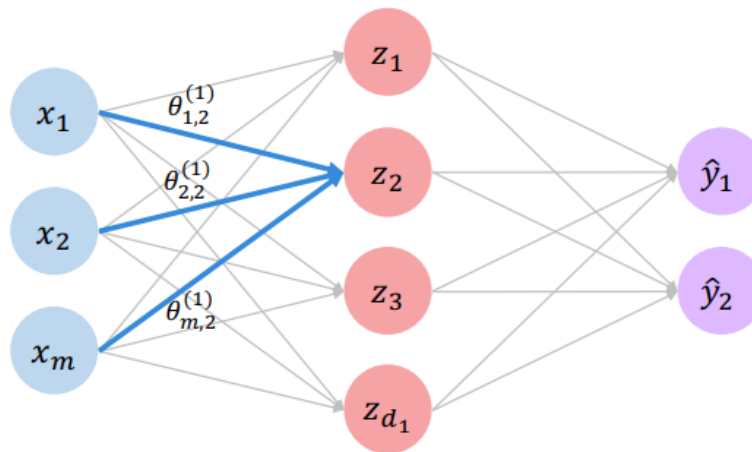
Linear Activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

Neural Network

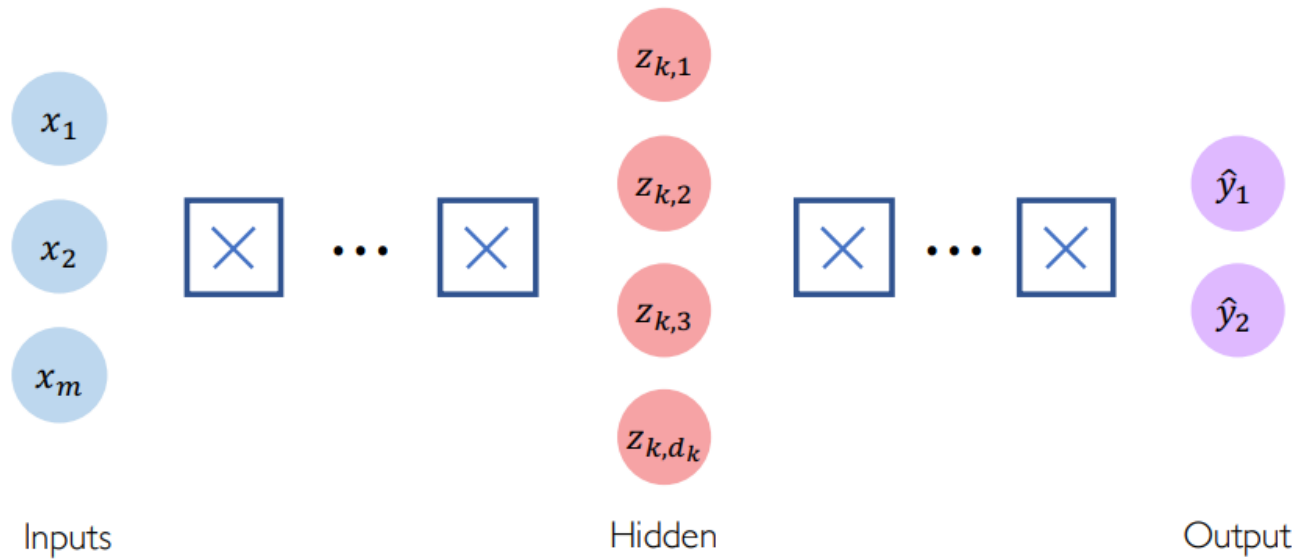
Single Layer Neural Network



$$\begin{aligned} z_2 &= \theta_{0,2}^{(1)} + \sum_{j=1}^m x_j \theta_{j,2}^{(1)} \\ &= \theta_{0,2}^{(1)} + x_1 \theta_{1,2}^{(1)} + x_2 \theta_{2,2}^{(1)} + x_m \theta_{m,2}^{(1)} \end{aligned}$$

Deep Neural Network

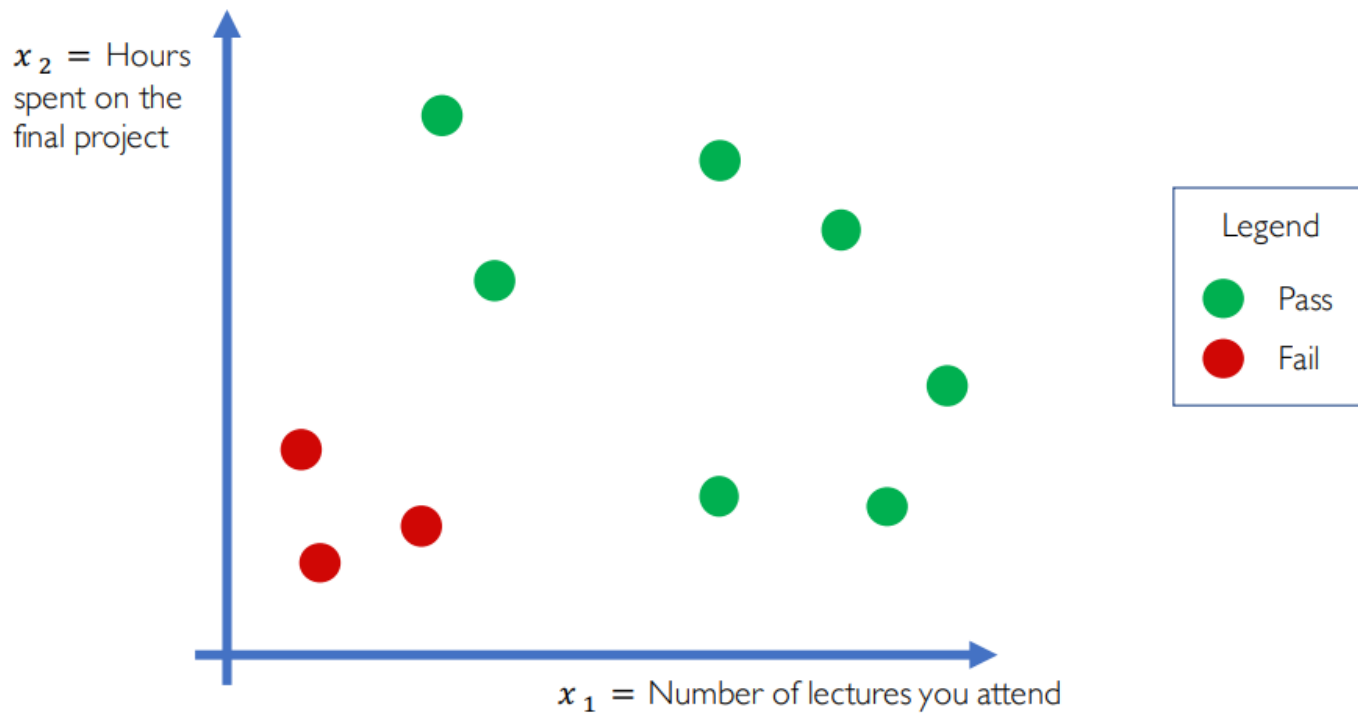
Deep Neural Network



$$z_{k,i} = \theta_{0,i}^{(k)} + \sum_{j=1}^{d_{k-1}} g(z_{k-1,j}) \theta_{j,i}^{(k)}$$

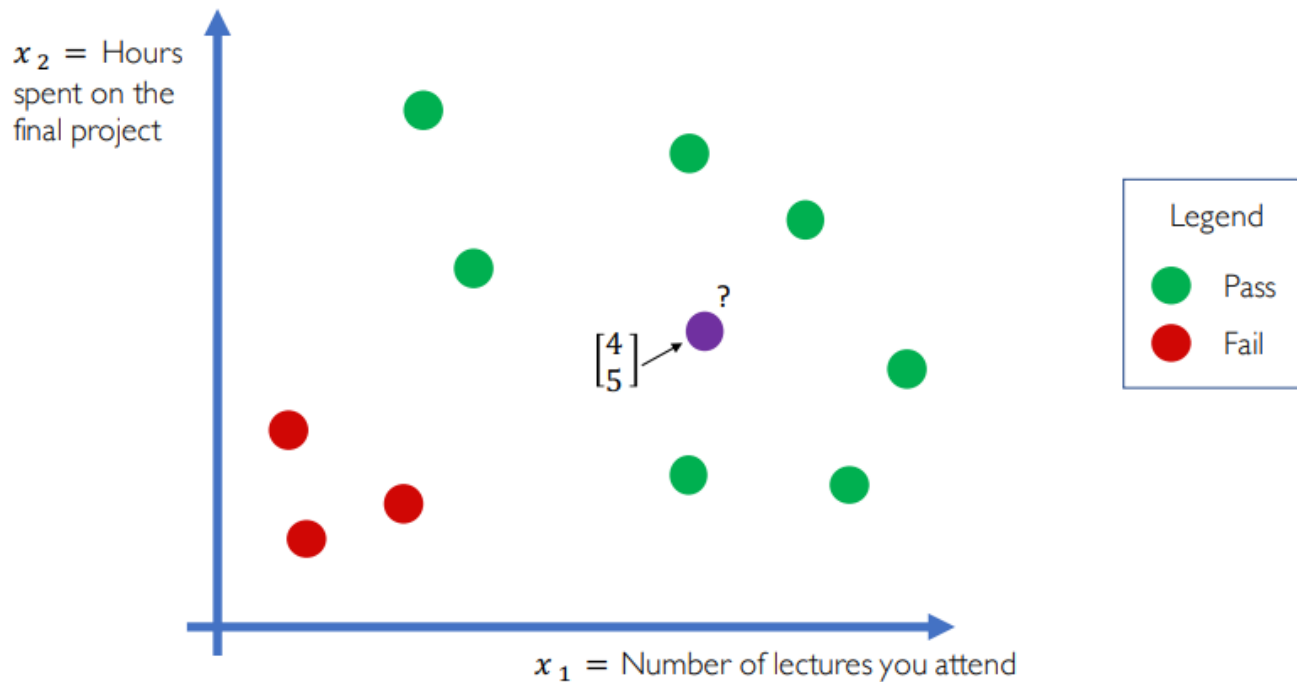
Example

Example Problem: Will I pass this class?



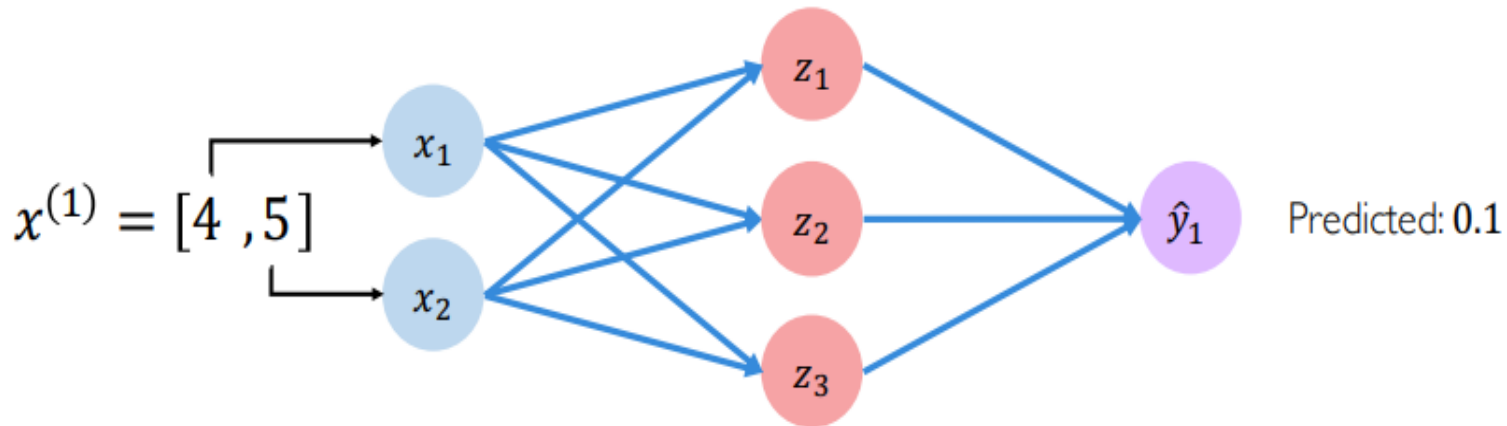
Example

Example Problem: Will I pass this class?



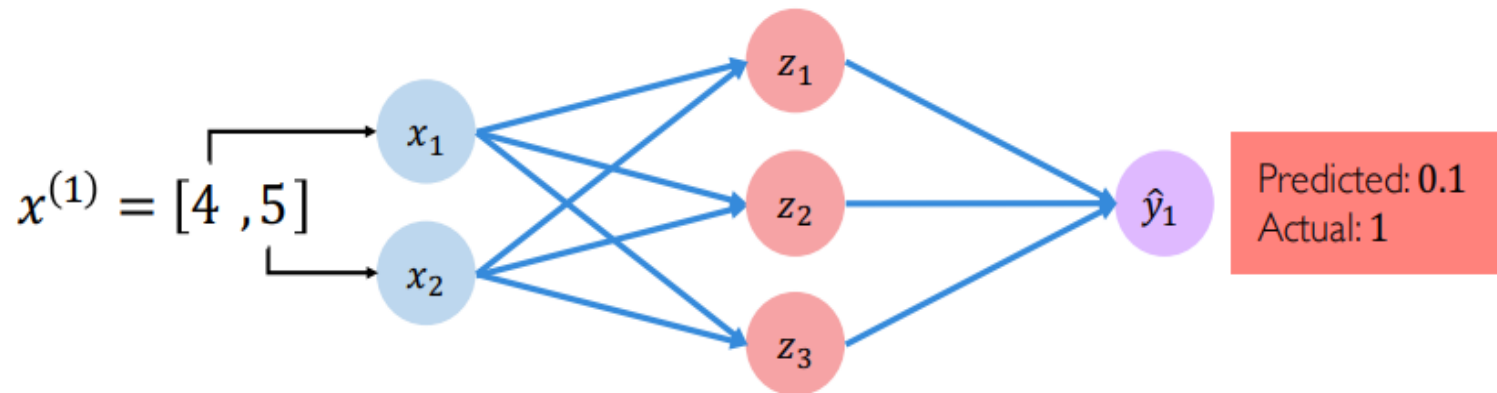
Example

Example Problem: Will I pass this class?



Example

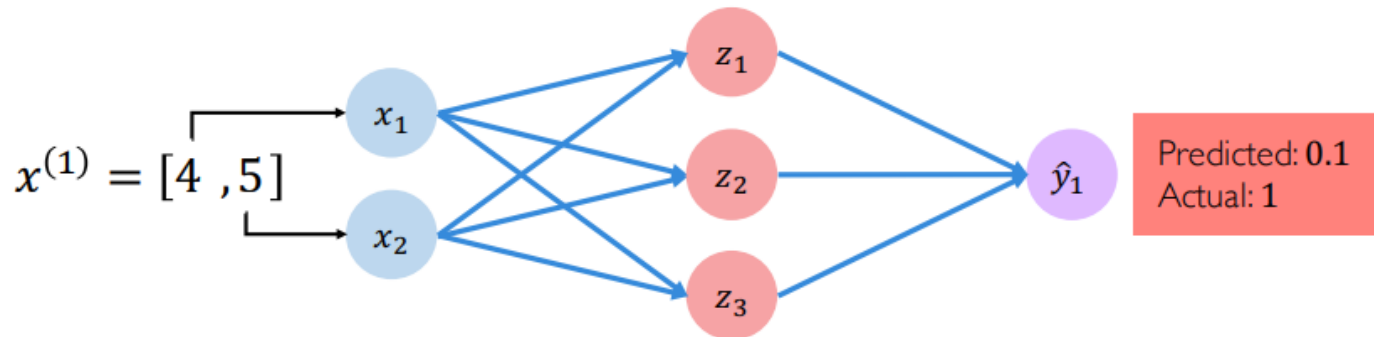
Example Problem: Will I pass this class?



Loss

Quantifying Loss

The **loss** of our network measures the cost incurred from incorrect predictions

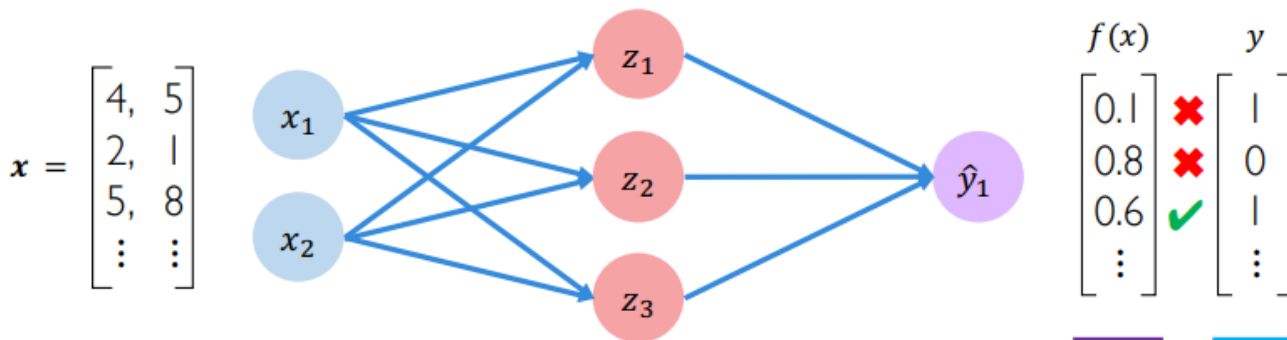


$$\mathcal{L}(\underbrace{f(x^{(i)}; \theta)}_{\text{Predicted}}, \underbrace{y^{(i)}}_{\text{Actual}})$$

Loss

Empirical Loss

The **empirical loss** measures the total loss over our entire dataset



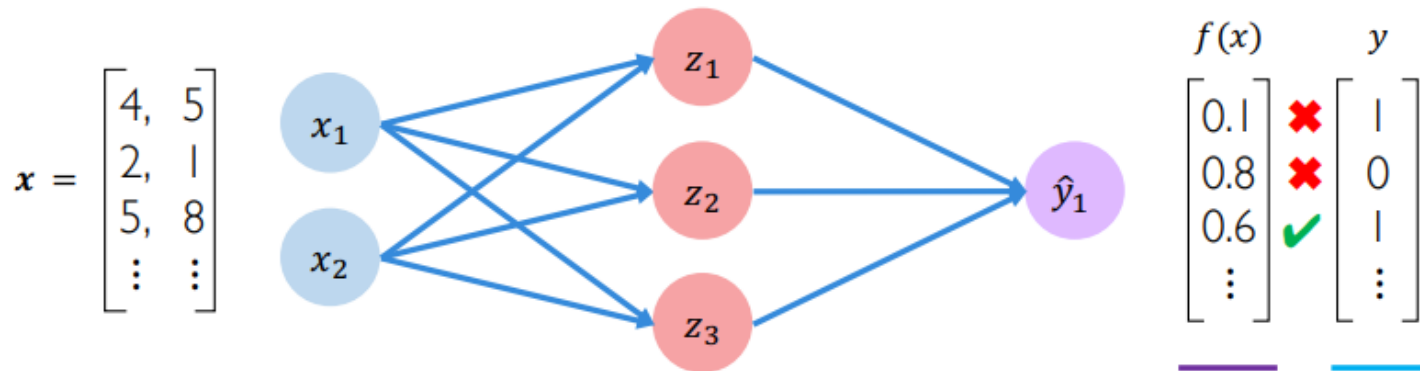
- Also known as:
- Objective function
 - Cost function
 - Empirical Risk

$$J(\theta) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}(\underbrace{f(x^{(i)}; \theta)}_{\text{Predicted}}, \underbrace{y^{(i)}}_{\text{Actual}})$$

Loss

Binary Cross Entropy Loss

Cross entropy loss can be used with models that output a probability between 0 and 1

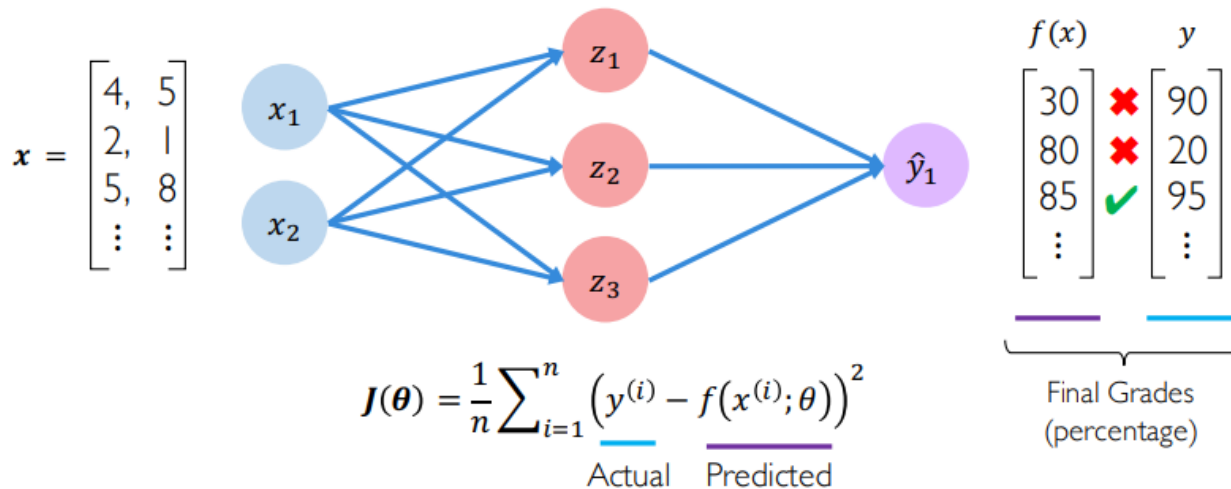


$$J(\theta) = \frac{1}{n} \sum_{i=1}^n \underbrace{y^{(i)}}_{\text{Actual}} \log \left(\underbrace{f(x^{(i)}; \theta)}_{\text{Predicted}} \right) + (1 - \underbrace{y^{(i)}}_{\text{Actual}}) \log \left(1 - \underbrace{f(x^{(i)}; \theta)}_{\text{Predicted}} \right)$$

Loss

Mean Squared Error Loss

Mean squared error loss can be used with regression models that output continuous real numbers



Training Neural Networks

Loss Optimization

We want to find the network weights that *achieve the lowest loss*

$$\theta^* = \operatorname{argmin}_{\theta} \frac{1}{n} \sum_{i=1}^n \mathcal{L}(f(x^{(i)}; \theta), y^{(i)})$$

$$\theta^* = \operatorname{argmin}_{\theta} J(\theta)$$

Training Neural Networks

Loss Functions Can Be Difficult to Optimize

Remember:

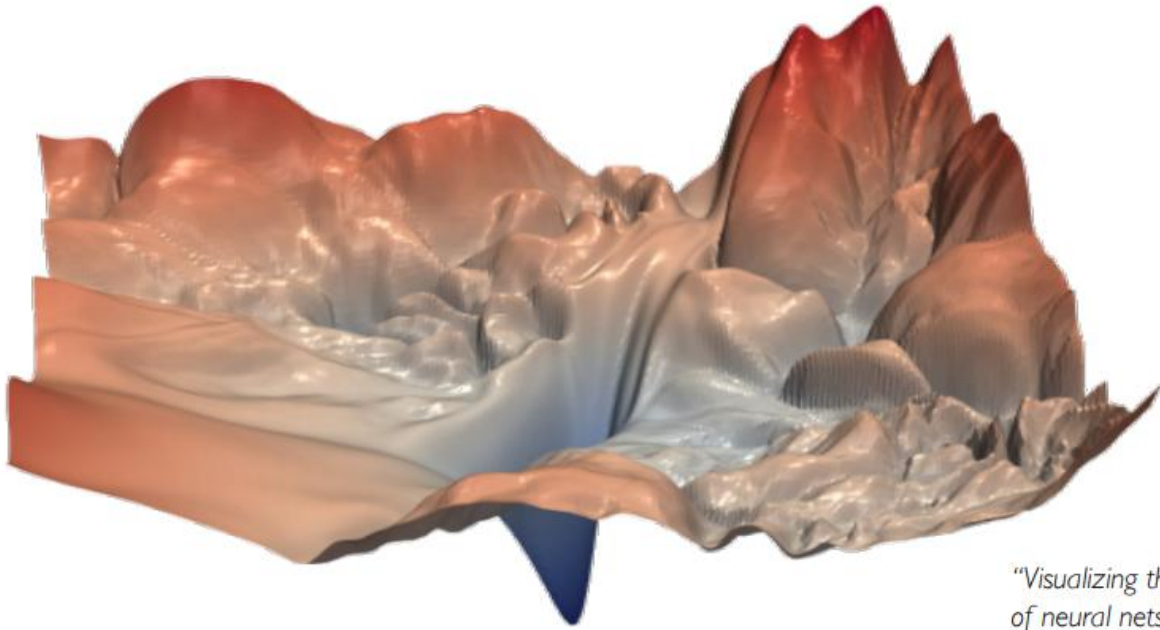
Optimization through gradient descent

$$\theta \leftarrow \theta - \eta \frac{\partial J(\theta)}{\partial \theta}$$

How can we set the
learning rate?

Training Neural Networks

Training Neural Networks is Difficult

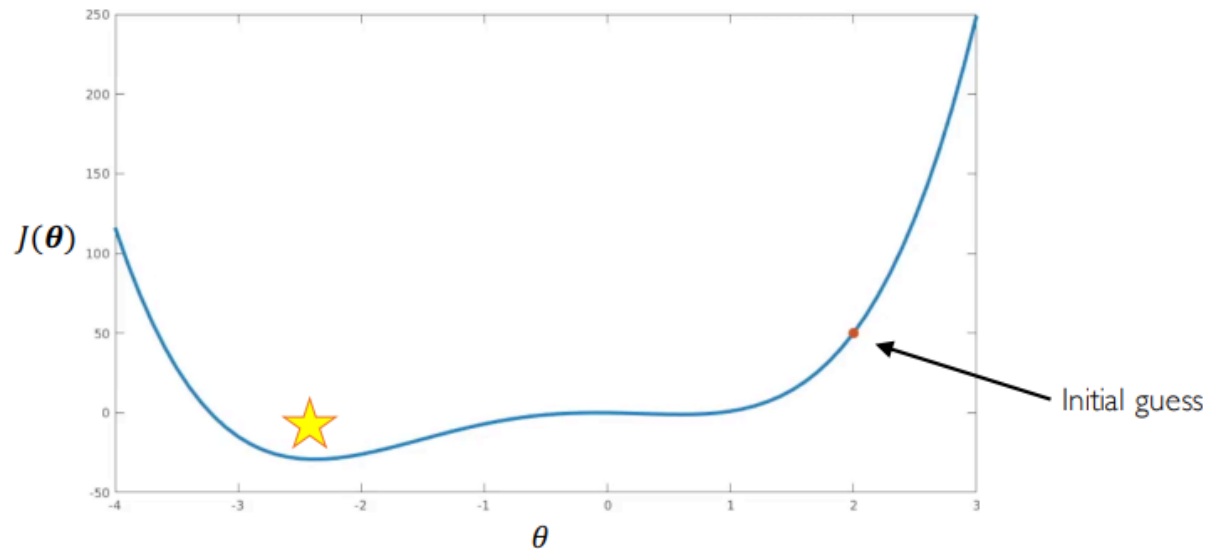


"Visualizing the loss landscape of neural nets". Dec 2017.

Learning Rate

Setting the Learning Rate

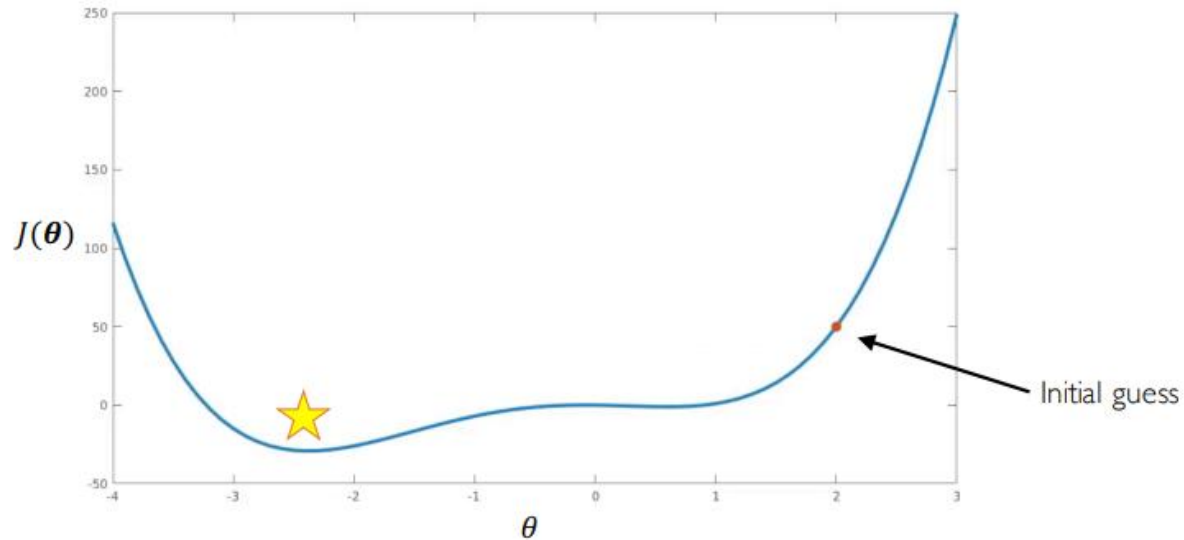
Small learning rate converges slowly and gets stuck in false local minima



Learning Rate

Setting the Learning Rate

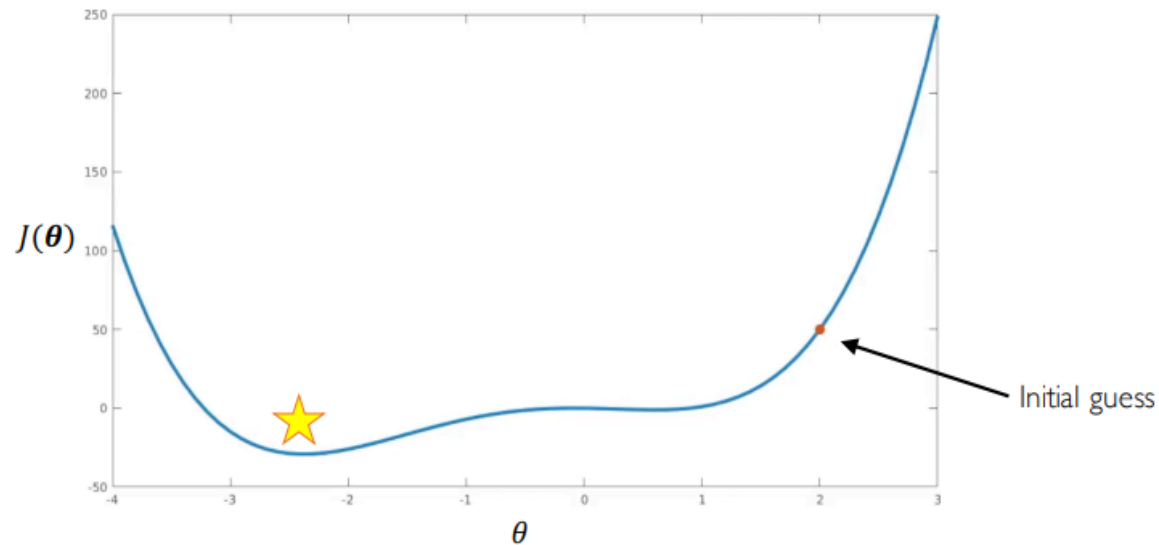
Large learning rates overshoot, become unstable and diverge



Learning Rate

Setting the Learning Rate

Stable learning rates converge smoothly and avoid local minima



Learning Rate

How to deal with this?

Idea 1:

Try lots of different learning rates and see what works “just right”

Or

Idea 2:

Do something smarter!
Design an adaptive learning rate that “adapts” to the landscape

Learning Rate

Adaptive Learning Rates

- Learning rates are no longer fixed
- Can be made larger or smaller depending on:
 - how large gradient is
 - how fast learning is happening
 - size of particular weights
 - etc...

Learning Rate

Adaptive Learning Rate Algorithms

- Momentum



`tf.train.MomentumOptimizer`

Qian et al. "On the momentum term in gradient descent learning algorithms." 1999.

- Adagrad



`tf.train.AdagradOptimizer`

Duchi et al. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." 2011.

- Adadelta



`tf.train.AdadeltaOptimizer`

Zeiler et al. "ADADELTA: An Adaptive Learning Rate Method." 2012.

- Adam



`tf.train.AdamOptimizer`

Kingma et al. "Adam: A Method for Stochastic Optimization." 2014.

- RMSProp



`tf.train.RMSPropOptimizer`

Mini-batches

Mini-batches while training

More accurate estimation of gradient

Smother convergence
Allows for larger learning rates

Mini-batches lead to fast training!

Can parallelize computation + achieve significant speed increases on GPU's

Regularization

Regularization

What is it?

Technique that constrains our optimization problem to discourage complex models

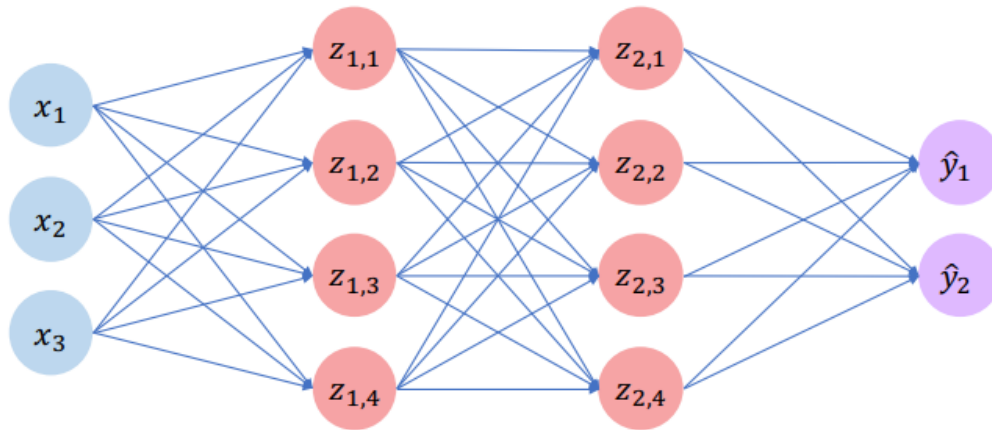
Why do we need it?

Improve generalization of our model on unseen data

Regularization

Regularization I: Dropout


- During training, randomly set some activations to 0

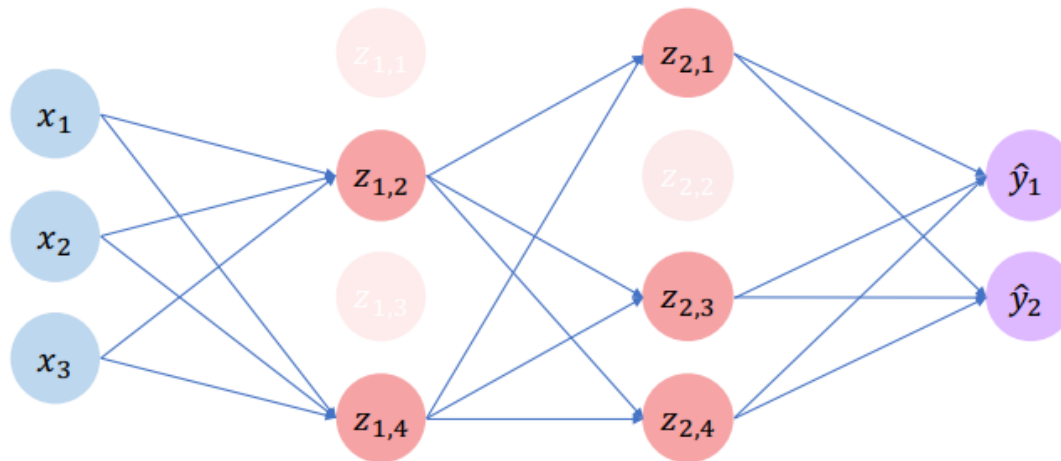


Regularization

Regularization I: Dropout

- During training, randomly set some activations to 0
 - Typically 'drop' 50% of activations in layer
 - Forces network to not rely on any 1 node


 `tf.nn.dropout(hiddenLayer, p=0.5)`

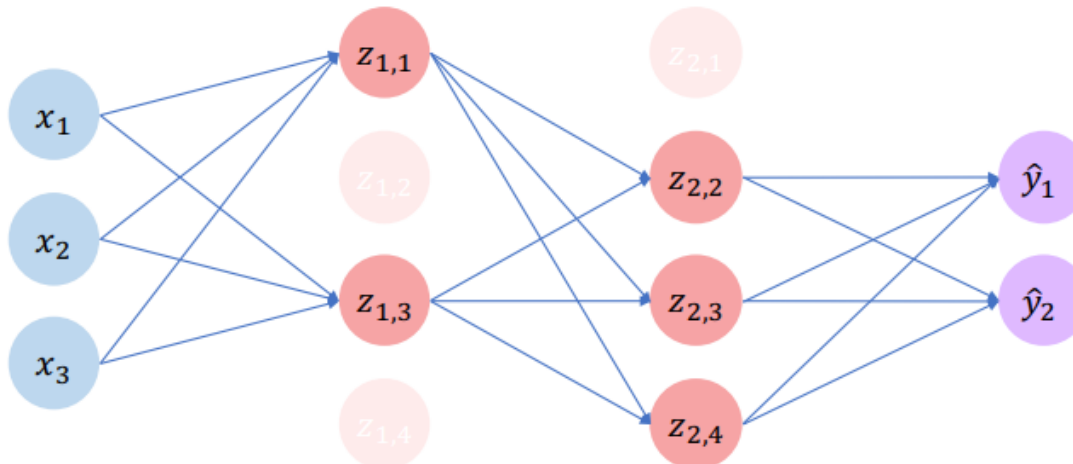


Regularization

Regularization I: Dropout

- During training, randomly set some activations to 0
 - Typically 'drop' 50% of activations in layer
 - Forces network to not rely on any 1 node

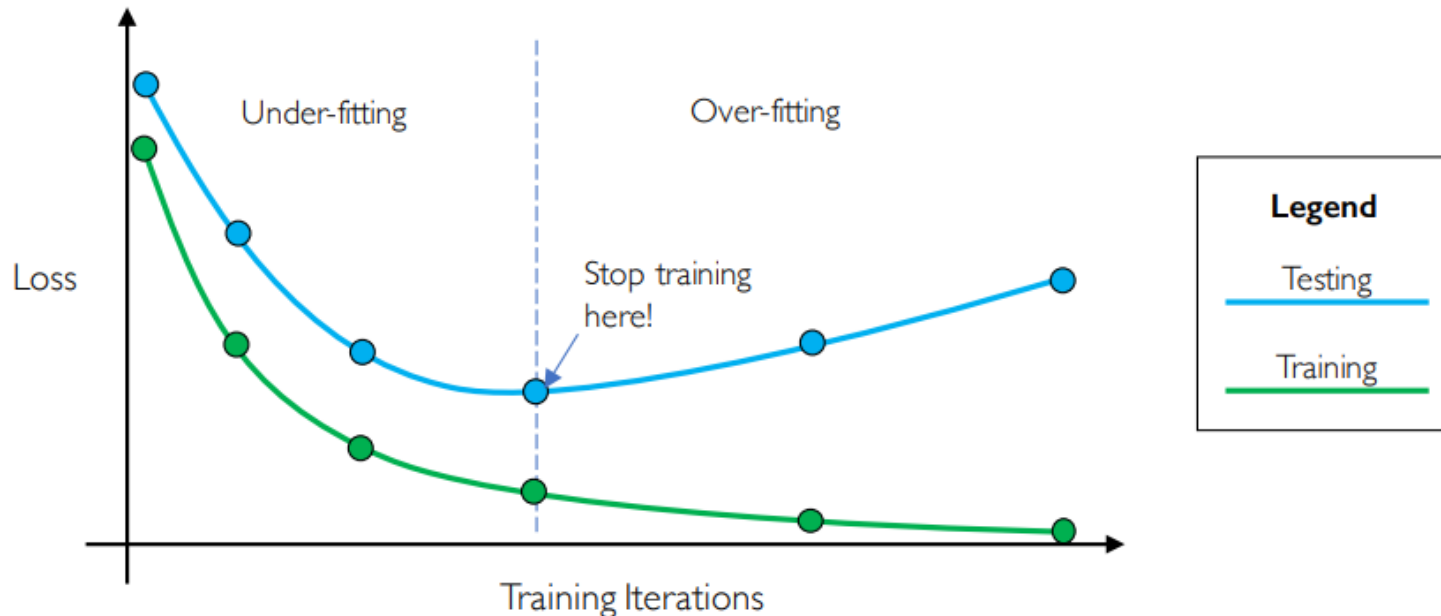
 `tf.nn.dropout(hiddenLayer, p=0.5)`



Regularization

Regularization 2: Early Stopping

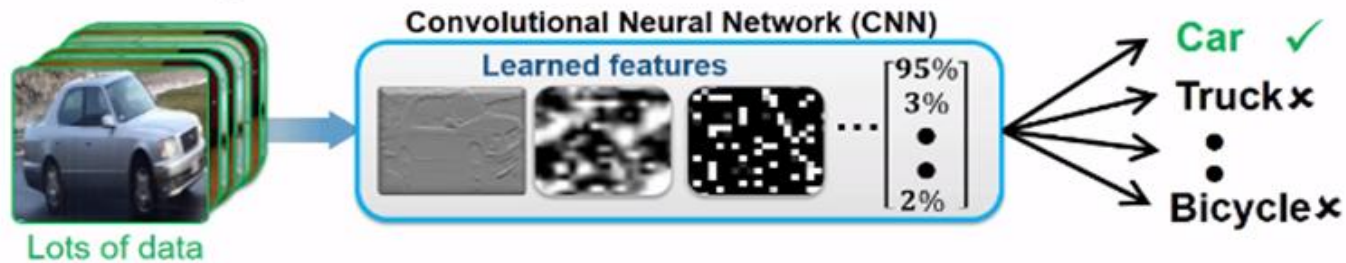
- Stop training before we have a chance to overfit



ConvNets

Approaches for Deep Learning

1. Train a Deep Neural Network from Scratch



2. Fine-tune a pre-trained model (transfer learning)



ConvNets

Deep Learning Approaches

Approach 1: Train a Deep Neural Network from Scratch

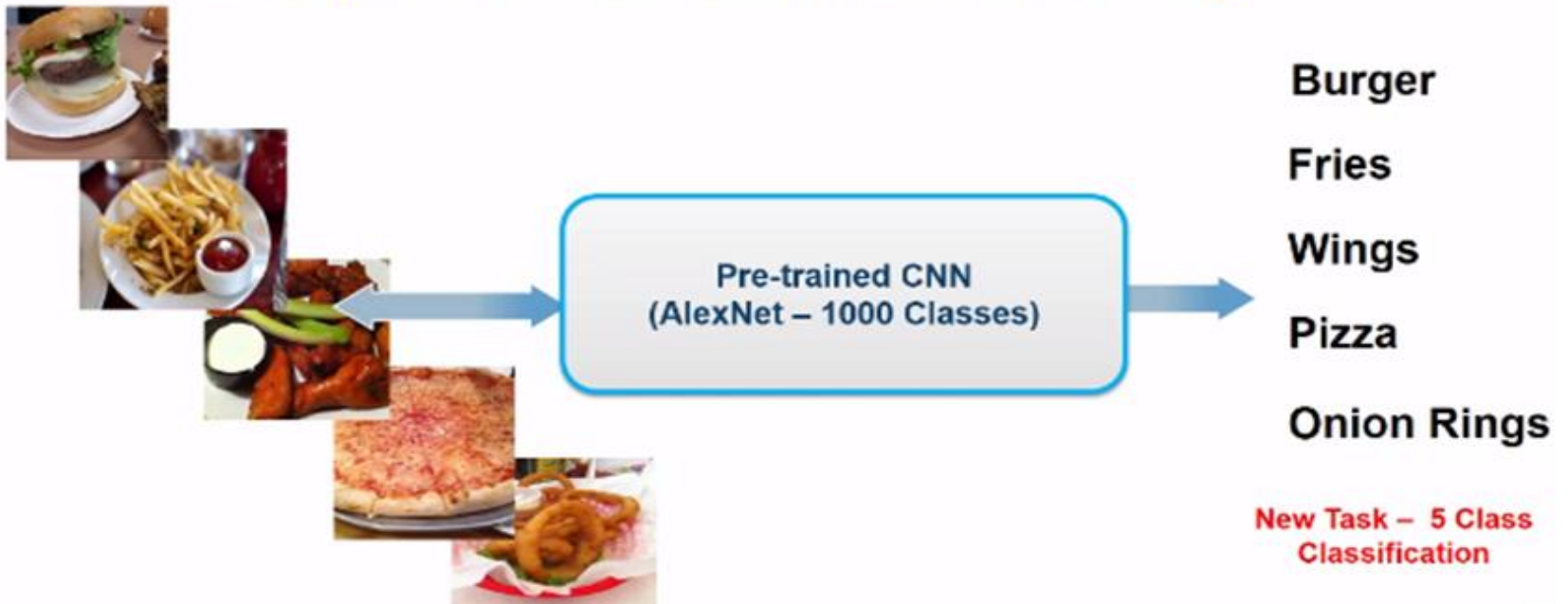


Recommended only when:

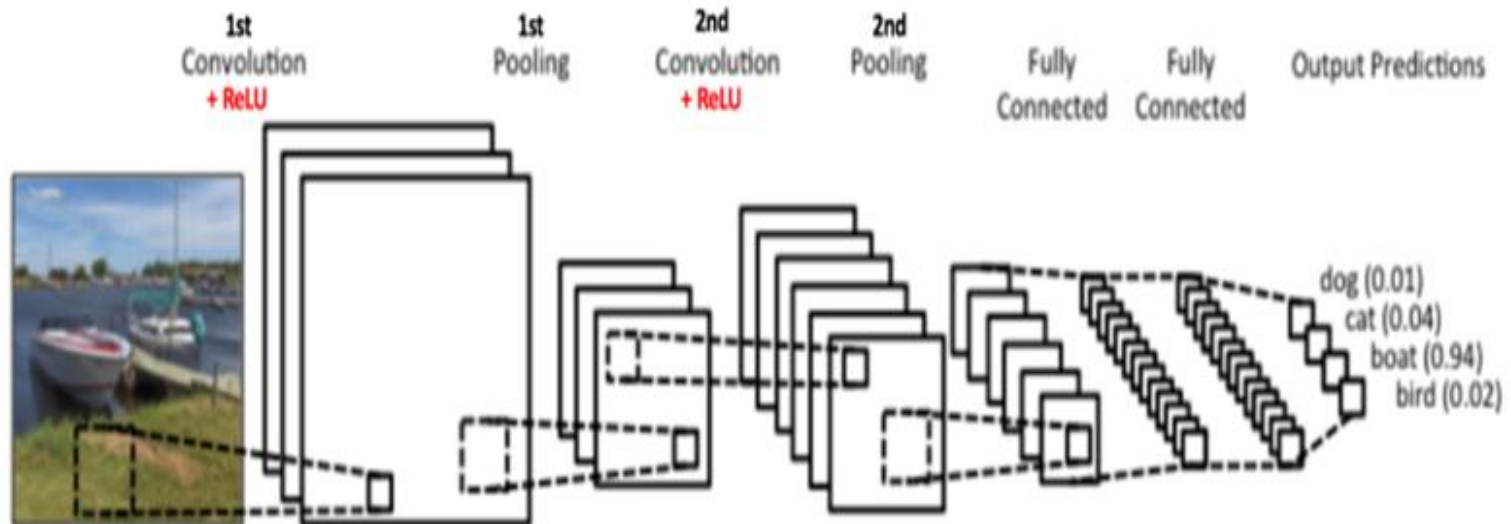
Training data	1000s to millions of labeled images
Computation	Compute intensive (requires GPU)
Training Time	Days to Weeks for real problems
Model accuracy	High (can over fit to small datasets)

ConvNets

Fine-tune a pre-trained model (Transfer learning)



ConvNets



Convolutional Layer

Convolution

Input Image

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Kernel

1	0	1
0	1	0
1	0	1

In CNN terminology, the 3×3 matrix is called a '**filter**' or 'kernel' or 'feature detector', and the matrix formed by sliding the filter over the image and computing the dot product is called the 'Convolved Feature' or 'Activation Map' or the '**Feature Map**'. It is important to note that filters act as feature detectors from the original input image.

1 _{x2}	1 _{x0}	1 _{x2}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x2}	0 _{x0}	1 _{x2}	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

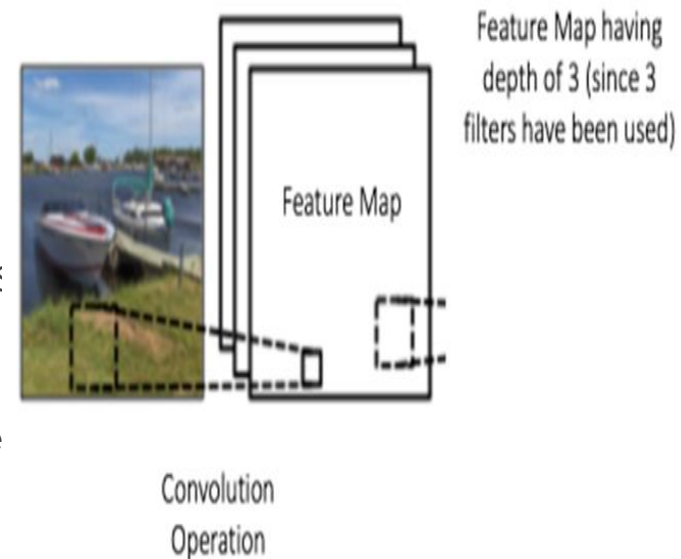
Convolved
Feature

Visit <https://cs231n.github.io/assets/conv-demo/index.html>

Feature Map Parameters

The size of the Feature Map (Convolved Feature) is controlled by three parameters that we need to decide before the convolution step is performed:

- **Depth:** Depth corresponds to the number of filters we use for the convolution operation. In the network shown in Figure, we are performing convolution of the original boat image using three distinct filters, thus producing three different feature maps as shown. You can think of these three feature maps as stacked 2d matrices, so, the 'depth' of the feature map would be three.



Feature Map Parameters

- **Stride:** Stride is the number of pixels by which we slide our filter matrix over the input matrix. When the stride is 1 we move the filters one pixel at a time. When the stride is 2, then the filters jump 2 pixels at a time as we slide them around. Having a larger stride will produce smaller feature maps.
- **Zero-padding:** Sometimes, it is convenient to pad the input matrix with zeros around the border, so that we can apply the filter to bordering elements of our input image matrix. A nice feature of zero padding is that it allows us to control the size of the feature maps. Adding zero-padding is also called wide convolution, and not using zero-padding would be a narrow convolution.

Size of Feature Map

Output width= $((W-F_w+2*P)/S)+1$

Output height= $((H-F_h+2*P)/S)+1$

Example

- ▶ Tensor size or shape: (width = 28, height = 28)
- ▶ Convolution filter size (F): (F_width = 5, F_height = 5)
- ▶ Padding (P): 0
- ▶ Stride (S): 1

Output width= $((28-5+2*0)/1)+1=24$

Output height= $((28-5+2*0)/1)+1=24$

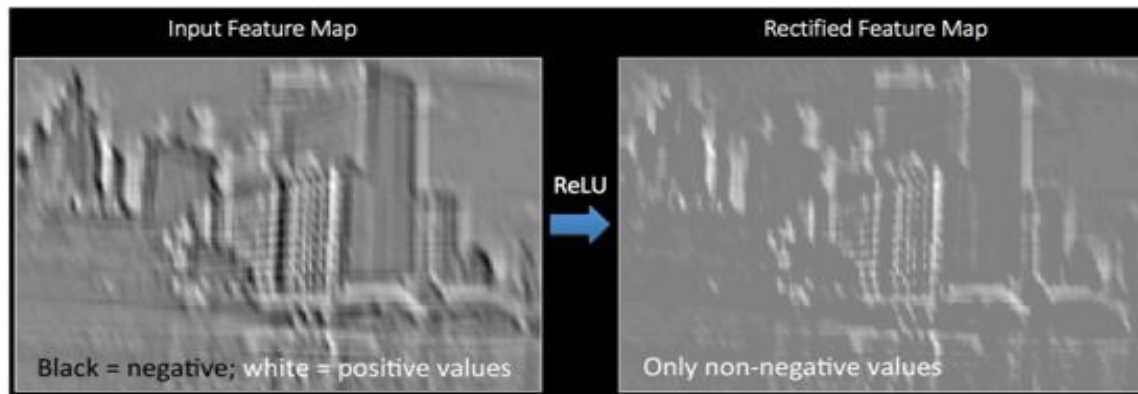
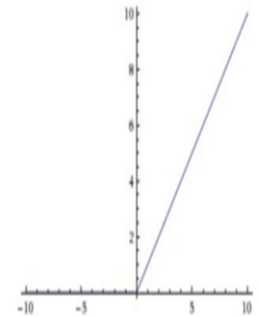
The output dimension will be (24,24)

Non Linearity (ReLU)

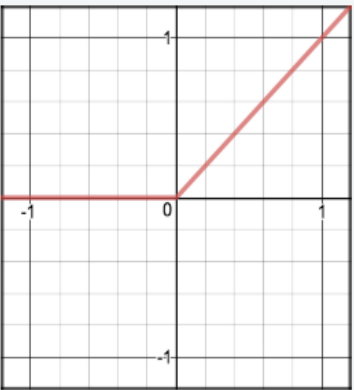
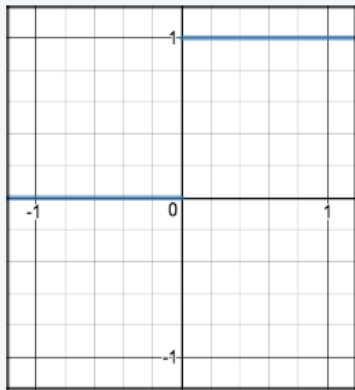
ReLU stands for Rectified Linear Unit and is a non-linear operation. Its output is given by:

ReLU is an element wise operation (applied per pixel) and replaces all negative pixel values in the feature map by zero. The purpose of ReLU is to introduce non-linearity in our ConvNet, since most of the real-world data we would want our ConvNet to learn would be non-linear (Convolution is a linear operation – element wise matrix multiplication and addition, so we account for non-linearity by introducing a non-linear function like ReLU).

$$\text{Output} = \text{Max}(\text{zero}, \text{Input})$$



Non Linearity (ReLU)

Function	Derivative
$R(z) = \begin{cases} z & z > 0 \\ 0 & z \leq 0 \end{cases}$	$R'(z) = \begin{cases} 1 & z > 0 \\ 0 & z < 0 \end{cases}$
	

Pros

- It avoids and rectifies vanishing gradient problem.
- ReLU is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations.

Cons

- One of its limitations is that it should only be used within Hidden layers of a Neural Network Model.
- Some gradients can be fragile during training and can die. It can cause a weight update which will make it never activate on any data point again. Simply saying that ReLU could result in Dead Neurons.
- In another words, for activations in the region ($x < 0$) of ReLU, gradient will be 0 because of which the weights will not get adjusted during descent. That means, those neurons which go into that state will stop responding to variations in error/ input (simply because gradient is 0, nothing changes). This is called dying ReLU problem.
- The range of ReLU is $[0, \infty)$. This means it can blow up the activation.

Leaky ReLU

LeakyRelu is a variant of ReLU. Instead of being 0 when $z < 0$, a leaky ReLU allows a small, non-zero, constant gradient α (Normally, $\alpha = 0.01$).

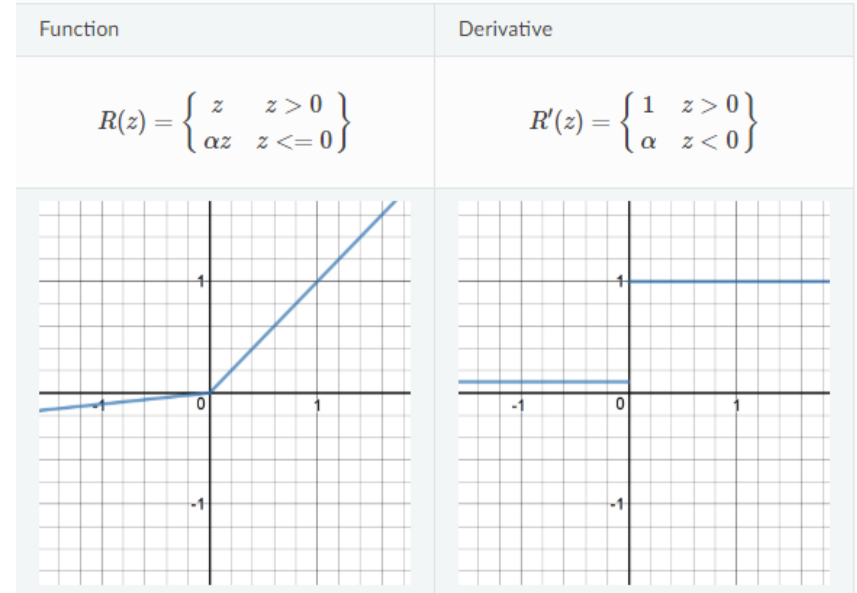
Pros

Leaky ReLUs are one attempt to fix the “dying ReLU” problem by having a small negative slope (of 0.01, or so).

Cons

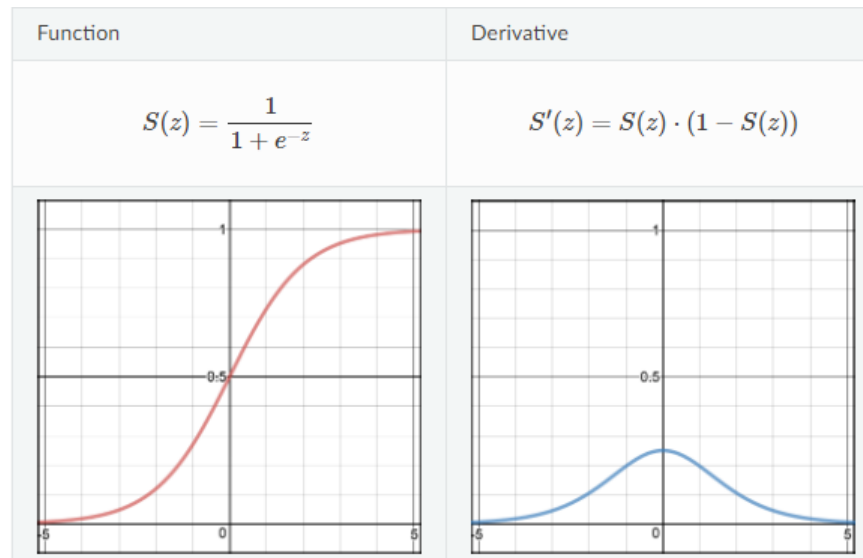
As it possess linearity, it can't be used for the complex Classification.

It lags behind the Sigmoid and Tanh for some of the use cases.



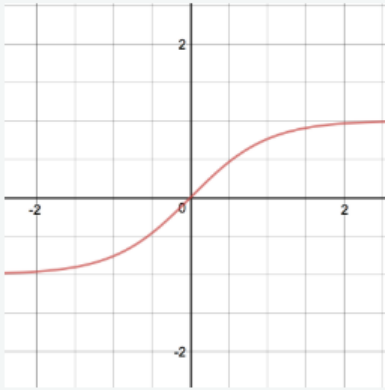
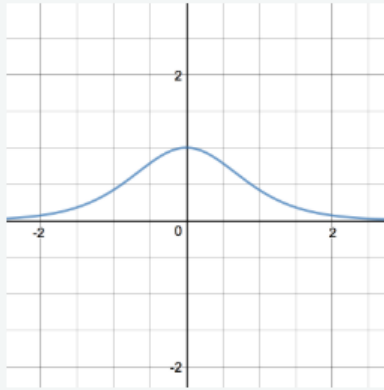
Other Activation Functions - Sigmoid

Sigmoid takes a real value as input and outputs another value between 0 and 1. It's easy to work with and has all the nice properties of activation functions: it's non-linear, continuously differentiable, monotonic, and has a fixed output range.



Other Activation Functions - Tanh

Tanh squashes a real-valued number to the range $[-1, 1]$. It's non-linear. But unlike Sigmoid, its output is zero-centered.

Function	Derivative
$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$\tanh'(z) = 1 - \tanh(z)^2$
 A graph of the tanh function on a coordinate plane. The x-axis ranges from -2 to 2, and the y-axis ranges from -2 to 2. The curve is red and passes through the origin (0,0). It approaches y = -1 as x goes to negative infinity and y = 1 as x goes to positive infinity.	 A graph of the derivative of the tanh function on a coordinate plane. The x-axis ranges from -2 to 2, and the y-axis ranges from -2 to 2. The curve is blue and is symmetric about the y-axis, with a peak at (0,1). It approaches y = 0 as x goes to positive or negative infinity.

Other Activation Functions - Softmax

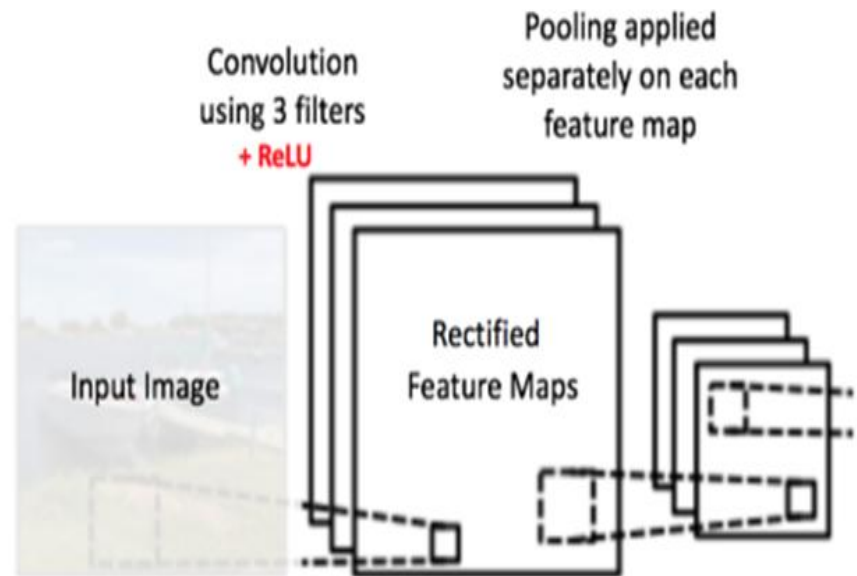
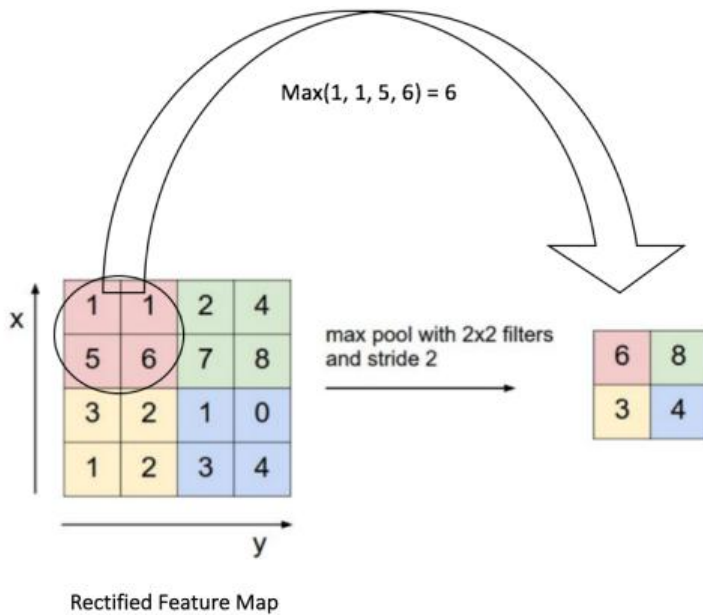
Softmax function calculates the probabilities distribution of the event over 'n' different events. In general way of saying, this function will calculate the probabilities of each target class over all possible target classes. Later the calculated probabilities will be helpful for determining the target class for the given inputs.

Pooling

Spatial Pooling (also called subsampling or downsampling) reduces the dimensionality of each feature map but retains the most important information. Spatial Pooling can be of different types: Max, Average, Sum etc.

In case of Max Pooling, we define a spatial neighborhood (for example, a 2×2 window) and take the largest element from the rectified feature map within that window. Instead of taking the largest element we could also take the average (Average Pooling) or sum of all elements in that window. In practice, Max Pooling has been shown to work better.

Pooling



Pooling

In particular, pooling:

- makes the input representations (feature dimension) smaller and more manageable
- reduces the number of parameters and computations in the network, therefore, controlling overfitting
- makes the network invariant to small transformations, distortions and translations in the input image (a small distortion in input will not change the output of Pooling – since we take the maximum / average value in a local neighborhood).
- helps us arrive at an almost scale invariant representation of our image (the exact term is “equivariant”). This is very powerful since we can detect objects in an image no matter where they are located

Padding

Padding can be Valid or Same

When padding == "VALID", the input image is not padded. This means that the filter window always stays inside the input image. This type of padding is called valid because for this padding only the valid and original elements of the input image are considered. When padding == "VALID", there can be a loss of information. Generally, elements on the right and the bottom of the image tend to be ignored. How many elements are ignored depends on the size of the kernel and the stride. In this case, the size of the output image \leq the size of the input image.

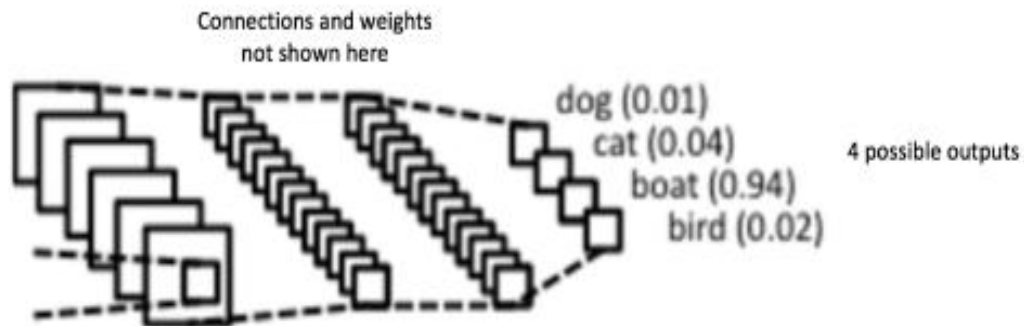
Padding

Padding can be Valid or Same

When padding == "SAME", the input is half padded. The padding type is called SAME because the output size is the same as the input size (when stride=1). Using 'SAME' ensures that the filter is applied to all the elements of the input. Normally, padding is set to "SAME" while training the model. Output size is mathematically convenient for further computation.

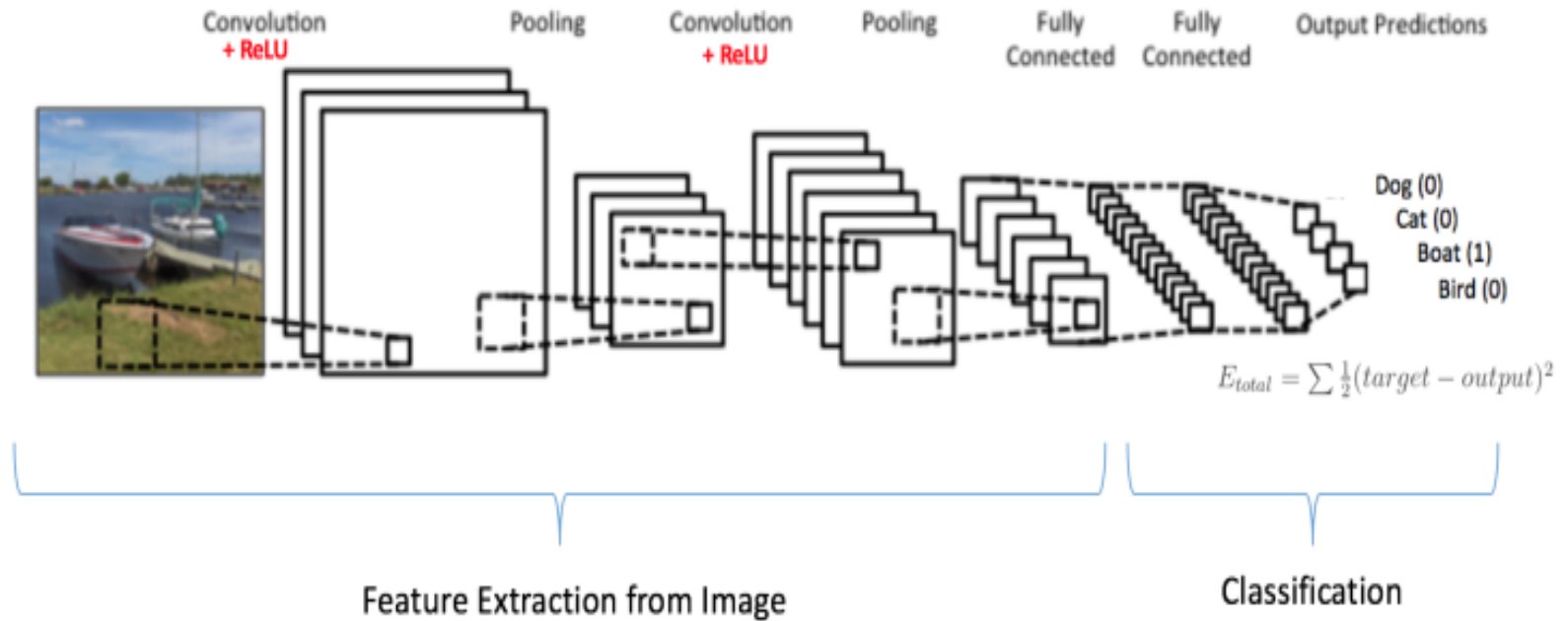
Fully Connected Layer

The Fully Connected layer is a traditional Multi Layer Perceptron that uses a softmax* activation function in the output layer (other classifiers like SVM can also be used). The term “Fully Connected” implies that every neuron in the previous layer is connected to every neuron on the next layer.



***Softmax classifiers give you *probabilities* for each class label.** Softmax classifier is a generalization of the binary form of Logistic Regression

The network



Training the network

The overall training process of the Convolution Network may be summarized as below:

Step1: We initialize all filters and parameters / weights with random values

Step2: The network takes a training image as input, goes through the forward propagation step (convolution, ReLU and pooling operations along with forward propagation in the Fully Connected layer) and finds the output probabilities for each class.

Step3: Calculate the total error at the output layer

Step4: Use Backpropagation to calculate the *gradients* of the error with respect to all weights in the network and use *gradient descent* to update all filter values / weights and parameter values to minimize the output error.

Step5: Repeat steps 2-4 with all images in the training set.

Testing the network

When a new (unseen) image is input into the ConvNet, the network would go through the forward propagation step and output a probability for each class (for a new image, the output probabilities are calculated using the weights which have been optimized to correctly classify all the previous training examples). If our training set is large enough, the network will (hopefully) generalize well to new images and classify them into correct categories.

Other layers - Batch Normalization layer

Batch normalization applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1. It is a process to make neural networks faster, more stable through adding extra layers in a deep neural network and reduce overfitting. The new layer performs the standardizing and normalizing operations on the input of a layer coming from a previous layer.

Importantly, batch normalization works differently during training and during inference.

During training the layer normalizes its output using the mean and standard deviation of the current batch of inputs.

During inference the layer normalizes its output using a moving average of the mean and standard deviation of the batches it has seen during training.

Defining the CNN architecture

The first step is to make a sheet containing the dimensions ,activation shapes and sizes for the architecture just as shown in the following example.

This computation tells us if we are choosing the right parameters while building our CNN architecture as the architecture must not end up with negative dimensions by going overboard with usage of high values of stride length and filter-sizes.

It is very important to ensure that we choose such values that there is a general decreasing trend in the Activation values and there aren't any very abrupt changes in Activation values.

Defining the activation shape

Consider the following example. The input image is 28x28x1

Layer	Number of Filters	Padding
Input Image	-	-
Conv2d(f=3,s=1)	8	Same
MaxPool(f=2,s=2)	-	Valid
Conv2d(f=5,s=1)	16	Valid
MaxPool(f=2,s=2)	-	Valid
Dense Flatten	-	-
Flatten	-	-
Dense	-	-
Softmax	-	-

Example taken from <https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii-hyper-parameter-42efca01e5d7>

Defining the activation shape

The activation shape after every convolution or pooling operation is

- $(\text{ceil}(N-f+1)/s, \text{ceil}(N-f+1)/s, \text{Number of filters})$ wherever the padding value is 'valid'
 - $(N, N, \text{Number of filters})$ where the padding used is 'same',
- where 'N' is input dimensions 'f' is filter size and 's' is stride's length.

Flatten makes the input into a one-dimensional list for input to the dense layers therefore Activation shape is (400,1).

Dense Activation shape (64,1) states 64 hidden units are used in the dense layer.

Layer	Number of Filters	Padding
Input Image	-	-
Conv2d(f=3,s=1)	8	Same
MaxPool(f=2,s=2)	-	Valid
Conv2d(f=5,s=1)	16	Valid
MaxPool(f=2,s=2)	-	Valid
Flatten	-	-
Flatten	-	-
Dense	-	-
Softmax	-	-



Layer	Number of Filters	Padding	Activation Shape
Input Image	-	-	(28,28,1)
Conv2d(f=3,s=1)	8	Same	(28,28,8)
MaxPool(f=2,s=2)	-	Valid	(14,14,8)
Conv2d(f=5,s=1)	16	Valid	(10,10,16)
MaxPool(f=2,s=2)	-	Valid	(5,5,16)
Flatten	-	-	(400,1)
Flatten	-	-	(120,0)
Dense	-	-	(64,1)
Softmax	-	-	(10,1)

Defining the activation size

The input layer's shape is (28, 28, 1), the activation size of that layer is $28 * 28 * 1 = 784$. The same holds good if you want to calculate the activation shape of any other layer. For example, if we want to calculate the activation size for CONV2 we just multiply (10,10,16) , i.e $10*10*16 = 1600$.

Layer	Number of Filters	Padding	Activation Shape	Activation Size
Input Image	-	-	(28,28,1)	784
Conv2d(f=3,s=1)	8	Same	(28,28,8)	6,272
MaxPool(f=2,s=2)	-	Valid	(14,14,8)	1568
Conv2d(f=5,s=1)	16	Valid	(8,8,16)	8300
MaxPool(f=2,s=2)	-	Valid	(5,5,16)	400
Flatten	-	-	(400,1)	400
Flatten	-	-	(120,0)	120
Dense	-	-	(64,1)	64
Softmax	-	-	(10,1)	10

Defining the number of parameters

The number of parameters per layer are computed by the following formula

$$\text{Parameters} = ((F_w * F_h * \text{number of filters in the previous layer} + 1) * \text{number of filters})$$

where F_w denotes the filter's width and F_h the filter's height.

Defining number of parameters

The **Input layer** has nothing to learn. It just provides the input image's shape. Therefore, there are no learnable parameters here. Thus, the number of **parameters is 0**.

The Pooling layer has no learnable parameters because all it does is calculate a specific number and no backpropagation learning is involved. Thus, the number of **parameters is 0**.

The Fully Connected Layer (FC) has learnable parameters. Actually, it has the largest number of parameters because every neuron is connected to every other neuron. The number of parameters in the **FC** are:
 $((\text{current layer neurons } c * \text{previous layer neurons } p) + 1 * c)$.

Defining number of parameters

1. The first layer (**input layer**) has no parameters.
 2. The number of parameters in the second layer **CONV1 (filter shape =3*3, stride=1) layer is:**
 $((\text{shape of width of filter} * \text{shape of height filter} * \text{number of filters in the previous layer}) + 1) * \text{number of filters}) = ((3 * 3 * 1) + 1) * 8 = 80.$
 3. The third layer (**Pooling layer**) has no parameters.
 4. The number of parameters in the fourth layer **CONV2(filter shape =5*5, stride=1) layer is:**
 $((\text{shape of width of filter} * \text{shape of height filter} * \text{number of filters in the previous layer}) + 1) * \text{number of filters}) = ((5 * 5 * 8) + 1) * 16 = 3216.$
 5. The fifth layer (Pooling layer) has no parameters.
 6. The sixth layer (Flatten layer) has no parameters
 7. The seventh layer (Dense) has $((120 * 400) + 120) = 48120$ parameters.
 8. The eighth layer (Dense) has $((84 * 120) + 84) = 10164$ parameters.
 9. The last layer has $((84 * 10) + 10) = 850$ parameters.
- Total Parameters: 62430

Layer	Number of Filters	Padding
Input Image	-	-
Conv2d(f=3,s=1)	8	Same
MaxPool(f=2,s=2)	-	Valid
Conv2d(f=5,s=1)	16	Valid
M Dense 2,s=2)	-	Valid
Flatten	-	-
Flatten	-	-
Dense	-	-
Softmax	-	-

Exercise

Given the following architecture, verify the activation shape, the activation size and the number of parameters.

SL.No		Activation Shape	Activation Size	# Parameters
1.	Input Layer:	(32, 32, 3)	3072	0
2.	CONV1 (f=5, s=1)	(28, 28, 8)	6272	608
3.	POOL1	(14, 14, 8)	1568	0
4.	CONV2 (f=5, s=1)	(10, 10, 16)	1600	3216
5.	POOL2	(5, 5, 16)	400	0
6.	FC3	(120, 1)	120	48120
7.	FC4	(84, 1)	84	10164
8.	<u>Softmax</u>	(10, 1)	10	850

Principles/Conventions to build a CNN architecture

The basic principle followed in building a convolutional neural network is to **'keep the feature space wide and shallow in the initial stages of the network, and the make it narrower and deeper towards the end.'** Keeping the above principle in mind we lay down a few conventions to be followed to guide you while building your CNN architecture

1. Always start by using smaller filters to collect as much local information as possible, and then gradually increase the filter width to reduce the generated feature space width to represent more global, high-level and representative information

2. Following the principle, the number of channels should be low in the beginning such that it detects low-level features which are combined to form many complex shapes (by increasing the number of channels) which help distinguish between classes.

The number of filters is increased to increase the depth of the feature space thus helping in learning more levels of global abstract structures. One more utility of making the feature space deeper and narrower is to shrink the feature space for input to the dense networks.

Principles/Conventions to build a CNN architecture

By convention the number of channels generally increase or stay the same while we progress through layers in our convolutional neural net architecture

3. General filter sizes used are 3x3, 5x5 and 7x7 for the convolutional layer for a moderate or small-sized images and for Max-Pooling parameters we use 2x2 or 3x3 filter sizes with a stride of 2. Larger filter sizes and strides may be used to shrink a large image to a moderate size and then go further with the convention stated.
4. Try using padding = same when you feel the border's of the image might be important or just to help elongate your network architecture as padding keeps the dimensions same even after the convolution operation and therefore you can perform more convolutions without shrinking size.
5. Keep adding layers until you over-fit. As once we achieved a considerable accuracy in our validation set we can use regularization components like l1/l2 regularization, dropout, batch norm, data augmentation etc. to reduce over-fitting
6. Always use classic networks like LeNet, AlexNet, VGG-16, VGG-19 etc. as an inspiration i.e. the trend in the layers Conv-Pool-Conv-Pool or Conv-Conv-Pool-Conv-Conv-Pool or the trend in the Number of channels 32-64-128 or 32-32-64-64 or trend in filter sizes, Max-pooling parameters etc.

Some Network Architectures

LeNet (1990s): Already covered here.

1990s to 2012: In the years from late 1990s to early 2010s convolutional neural network were in incubation. As more and more data and computing power became available, tasks that convolutional neural networks could tackle became more and more interesting.

AlexNet (2012) – In 2012, Alex Krizhevsky (and others) released [AlexNet](#) which was a deeper and much wider version of the LeNet and won by a large margin the difficult ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012. It was a significant breakthrough with respect to the previous approaches and the current widespread application of CNNs can be attributed to this work.

ZF Net (2013) – The ILSVRC 2013 winner was a Convolutional Network from Matthew Zeiler and Rob Fergus. It became known as the [ZFNet](#) (short for Zeiler & Fergus Net). It was an improvement on AlexNet by tweaking the architecture hyperparameters.

Some Network Architectures

GoogLeNet (2014) – The ILSVRC 2014 winner was a Convolutional Network from [Szegedy et al.](#) from Google. Its main contribution was the development of an *Inception Module* that dramatically reduced the number of parameters in the network (4M, compared to AlexNet with 60M).

VGGNet (2014) – The runner-up in ILSVRC 2014 was the network that became known as the [VGGNet](#). Its main contribution was in showing that the depth of the network (number of layers) is a critical component for good performance.

Some Network Architectures

ResNets (2015) – Residual Network developed by Kaiming He (and others) was the winner of ILSVRC 2015. ResNets are currently by far state of the art Convolutional Neural Network models and are the default choice for using ConvNets in practice (as of May 2016).

DenseNet (August 2016) – Recently published by Gao Huang (and others), the Densely Connected Convolutional Network has each layer directly connected to every other layer in a feed-forward fashion. The DenseNet has been shown to obtain significant improvements over previous state-of-the-art architectures on five highly competitive object recognition benchmark tasks.

Some Network Architectures - Parameters

Network	Number of Parameters
LeNet-5	60000
AlexNet	60 million
VGG-16	138 million
GoogleNet	5 million (V1) - 23 million (V3)
ResNet50	25 million
DenseNet-190	40 million
KarNet	16 million

Transfer learning

Transfer learning is about leveraging feature representations from a pre-trained model, so you don't have to train a new model from scratch.

The pre-trained models are usually trained on massive datasets that are a standard benchmark in the computer vision frontier. The weights obtained from the models can be reused in other computer vision tasks.

Transfer learning

These models can be used directly in making predictions on new tasks or integrated into the process of training a new model.

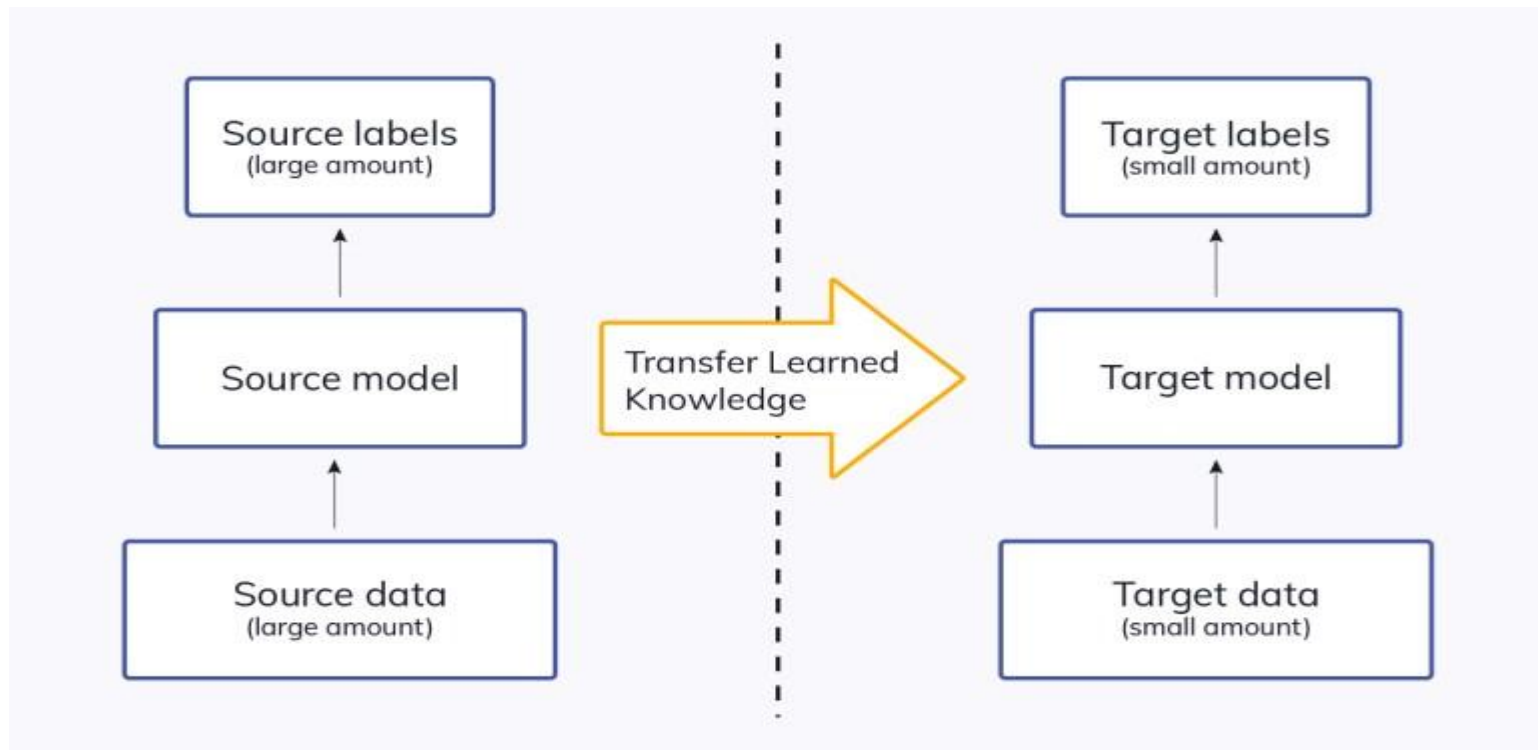
Including the pre-trained models in a new model leads to:

- lower training time
- lower generalization error.

Transfer learning is particularly very useful when you have a **small training dataset**. In this case, you can, for example, use the weights from the pre-trained models to **initialize the weights** of the new model.

Transfer learning can also be applied to natural language processing problems.

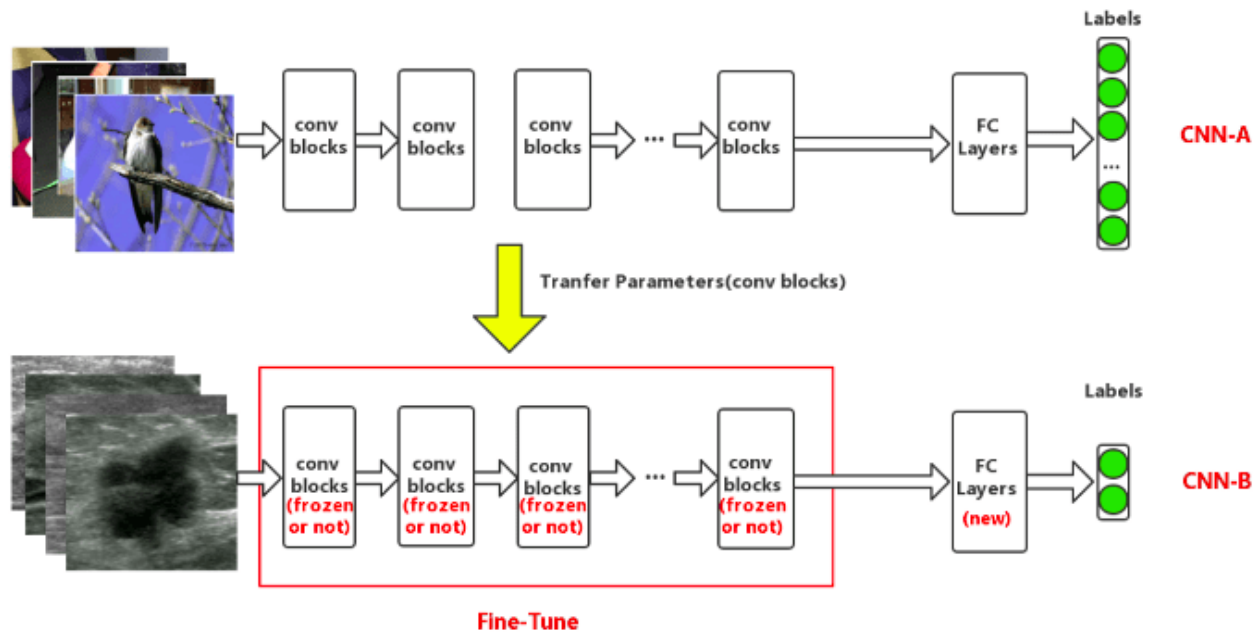
Transfer learning



Source : <https://neptune.ai/blog/transfer-learning-guide-examples-for-images-and-text-in-keras>

Transfer learning – Whats next ?

Fine-tuning is an optional step in transfer learning. Fine-tuning will usually improve the performance of the model. However, since you have to retrain the entire model, you'll likely overfit.



Transfer learning – What's next ?

Overfitting is avoidable. To limit overfitting:

- Retrain the model or part of it using a **low learning rate**. This is important because it prevents significant updates to the gradient. These updates result in poor performance.
- Use a callback to stop the training process when the model has stopped improving is also helpful.
- Use a dropout layer

Transfer learning – A case study

Assume you have 100 images of cats and 100 dogs and want to build a model to classify the images. How would you train a model using this small dataset? You can train your model from scratch, but it will most likely overfit horribly.

In general you want to use transfer learning because:

- **training models with high accuracy requires a lot of data.** For example, the ImageNet dataset contains over 1 million images. In the real world, you are unlikely to have such a large dataset.
- assuming that you had that kind of dataset, you might still **not have the resources required to train a model** on such a large dataset. Hence transfer learning makes a lot of sense if you don't have the compute resources needed to train models on huge datasets.
- even if you had the computational resources at your disposal, you still have to **wait for days or weeks to train such a model.** Therefore, using a pre-trained model will save you precious time.

When does transfer learning not work?

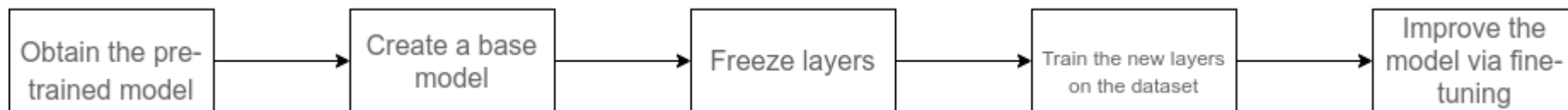
Transfer learning will not work when the high-level features learned by the bottom layers are not sufficient to differentiate the classes in your problem.

For example, a pre-trained model may be very good at identifying a door but not whether a door is closed or open. In this case, you can use the low-level features (of the pre-trained network) instead of the high-level features. In this case, you will have to retrain more layers of the model or use features from earlier layers.

When datasets are not similar, features transfer poorly. This [paper](#) investigates the similarity of datasets in more detail. That said, as shown in the paper, initializing the network with pre-trained weights results in better performance than using random weights.

Transfer learning - Implementation

You can implement transfer learning in these five general steps



Transfer learning - Implementation

1. Obtain the pre-trained model
 - Keras pre-trained [models](#) .
 - [TensorFlow Hub](#)

For text classification problems visit [Glove](#) , [Google's Word2vec](#) , [Hugging Face](#)

Transfer learning - Implementation

2. Create a base model

The first step is to instantiate the **base model** using one of the architectures such as ResNet or Xception. You can also optionally download the **pre-trained weights**.

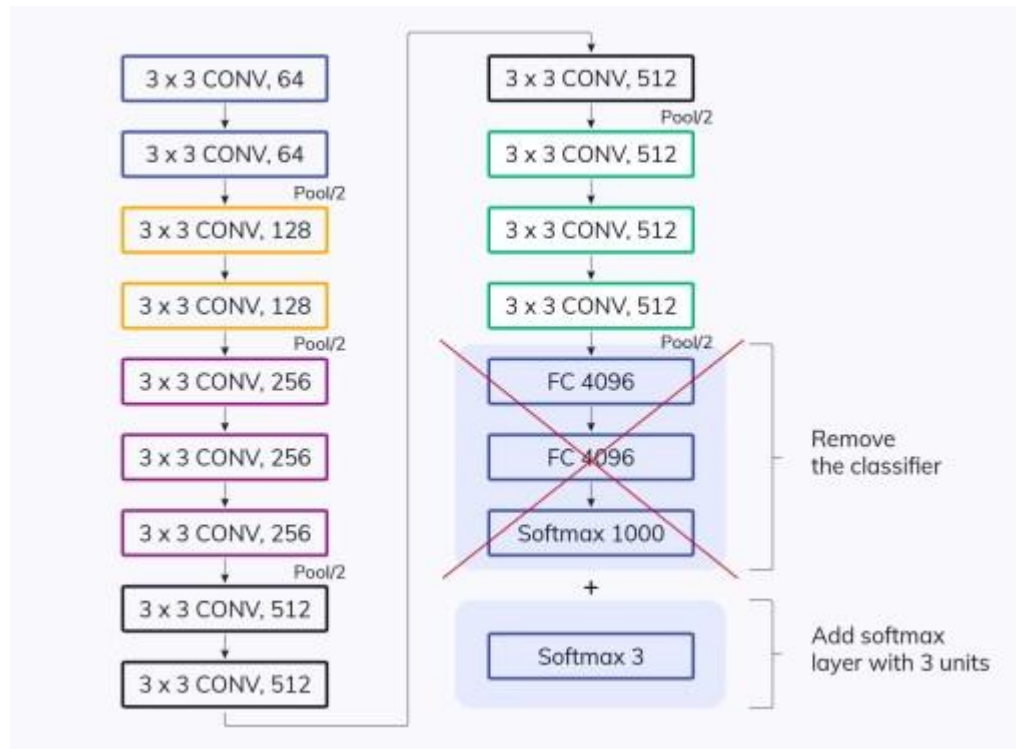
If you don't download the weights, you will have to use the architecture to train your model from scratch.

Recall that the base model will usually have more units in the final output layer than you require.

When creating the base model, you, therefore, have to remove the final output layer. Later on, you will add a final output layer that is compatible with your problem.

Transfer learning - Implementation

2. Create a base model



Transfer learning - Implementation

3. Freeze layers

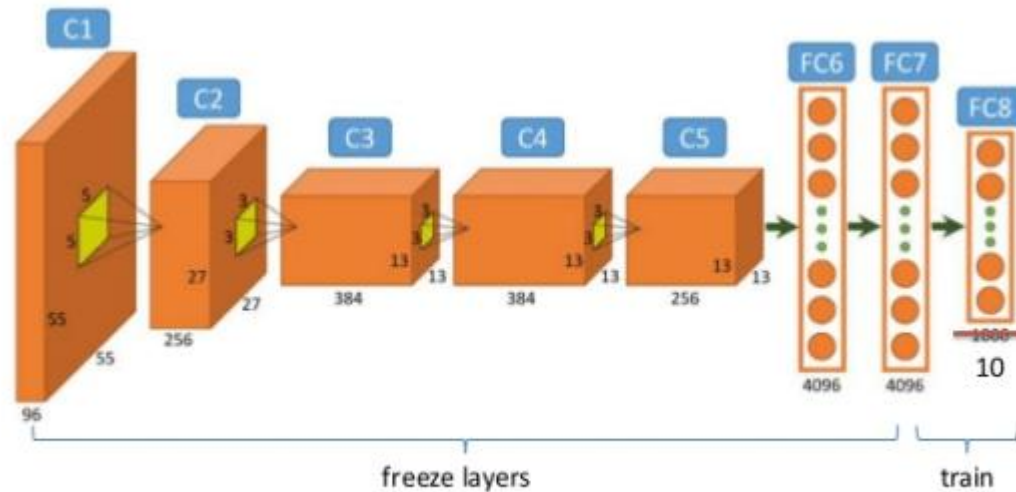
Freezing the layers from the pre-trained model is vital. This is because you don't want the **weights in those layers to be re-initialized**.

If they are, then you will lose all the learning that has already taken place. This will be no different from training the model from scratch.

```
base_model.trainable = False
```

Transfer learning - Implementation

3. Freeze layers



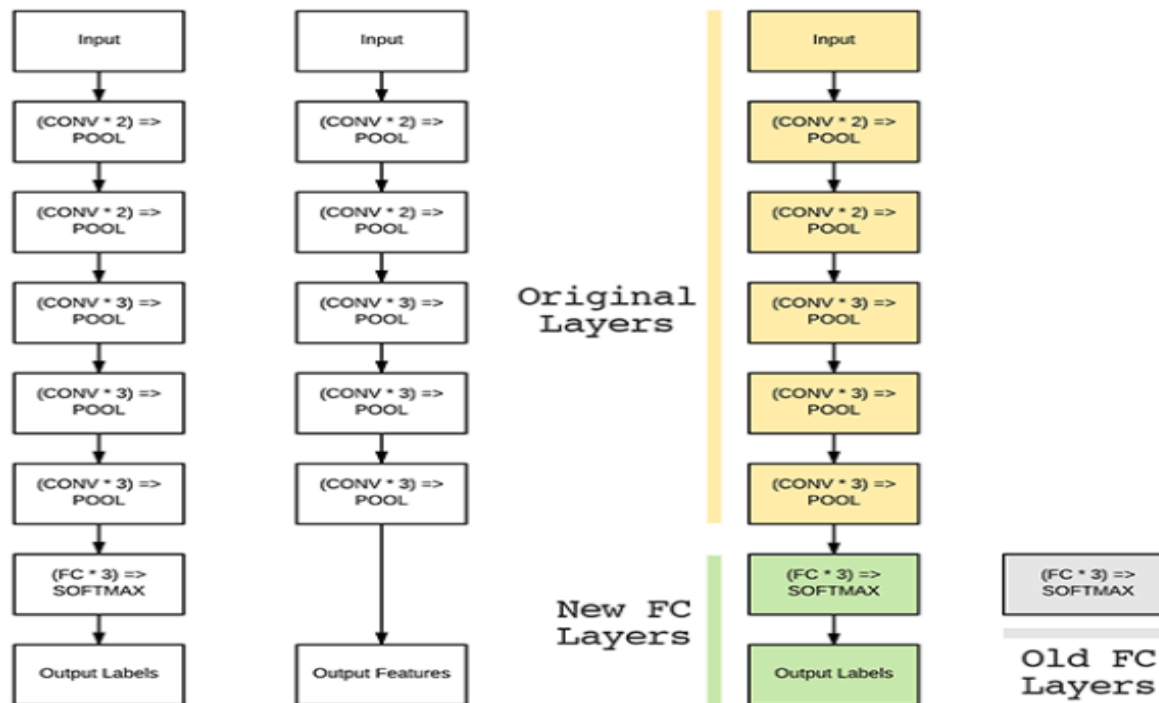
Transfer learning - Implementation

4. Add new trainable layers - Train the new layers

The next step is to **add new trainable layers** that will turn old features into predictions on the new dataset. This is important because the pre-trained model is loaded without the final output layer.

Transfer learning - Implementation

4. Add new trainable layers – Train the new layers



Transfer learning - Implementation

4. Add new trainable layers – Train the new layers

The pre-trained model's final output will most likely be different from the output that you want for your model.

For example, pre-trained models trained on the ImageNet dataset will output 1000 classes. However, your model might just have two classes. In this case, you have to train the model with a new output layer in place.

Therefore, you will add some new dense layers as you please, but most importantly, a final dense layer with **units corresponding to the number of outputs expected by your model.**

Transfer learning - Implementation

5. Improve the model via fine-tuning

Fine-tuning is done by unfreezing the base model or part of it and training the entire model again on the whole dataset at a very low learning rate.

The low learning rate will increase the performance of the model on the new dataset while preventing overfitting.

The learning rate has to be low because the model is quite large while the dataset is small.

Transfer learning - Implementation

5. Improve the model via fine-tuning

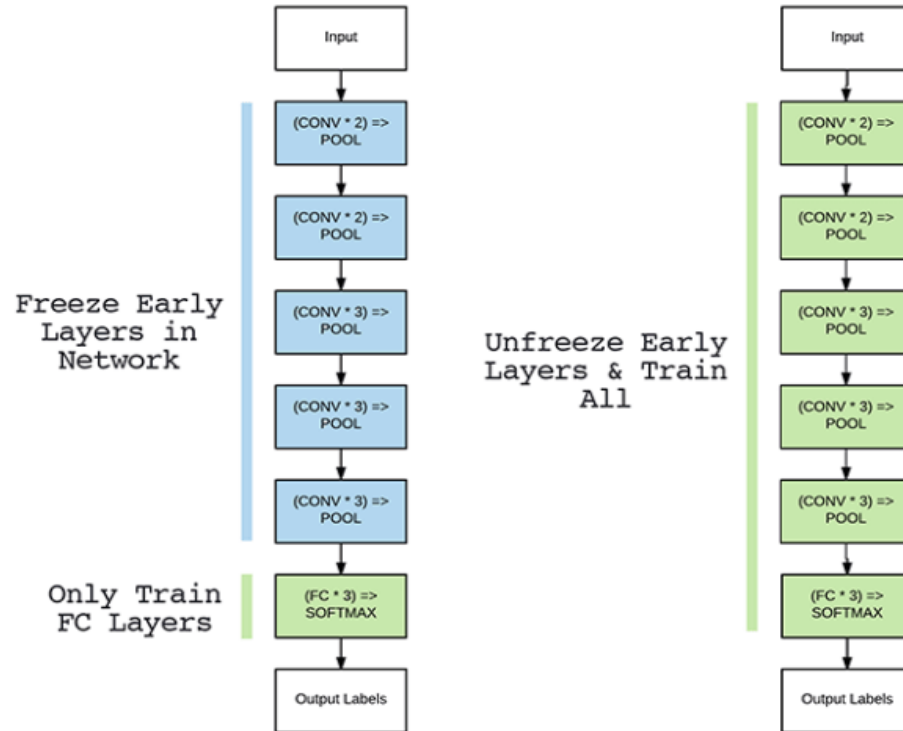
Recompile the model once you have made these changes so that they can take effect.

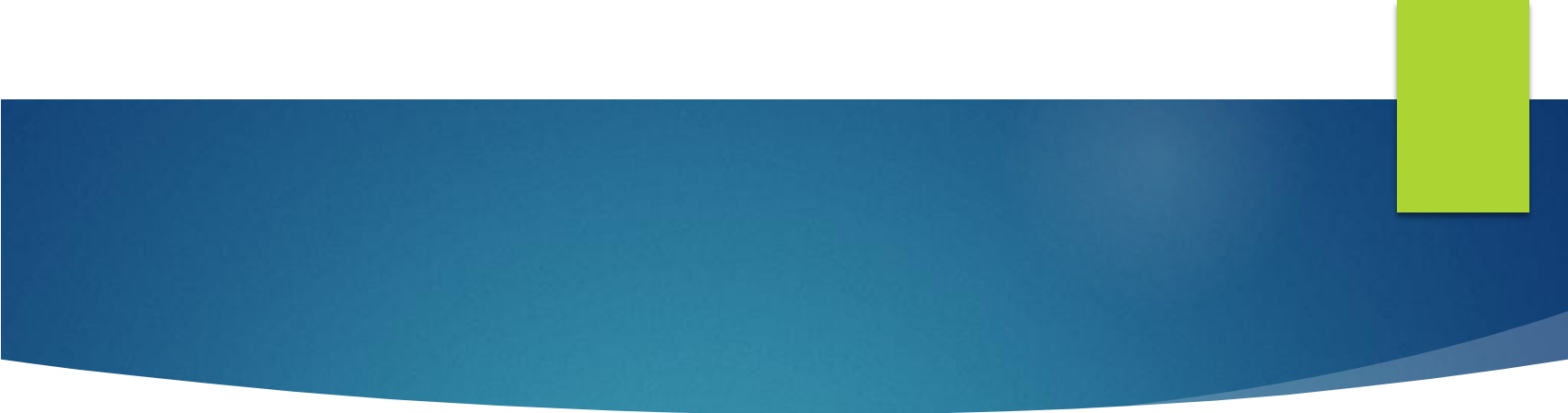
This is because the behavior of a model is frozen whenever you call the compile function. That means that you have to call the compile function again whenever you want to change the model's behavior.

The next step will be to train the model again while monitoring it via callbacks to ensure it does not overfit.

Transfer learning - Implementation

5. Improve the model via fine-tuning



- 
- Long Short-Term Memory (LSTM)
 - Generative Adversarial Networks (GANs)
 - Autoencoders
 - Restricted Boltzmann Machine (RBM)
 - Deep Belief Networks (DBN)

Μηχανική Μάθηση & Εξόρυξη Γνώσης

Ερωτήσεις
?

Βιβλιογραφία

- ▶ Ι. Βλαχάβας, Π. Κεφαλάς, Ν. Βασιλειάδης, Φ. Κόκκορας, Η. Σακελλαρίου, Τεχνητή Νοημοσύνη - Γ' Έκδοση, ISBN: 978-960-8396-64-7, Έκδοση/Διάθεση: Εκδόσεις Πανεπιστημίου Μακεδονίας, 2011
- ▶ Ian H. Witten and Eibe Frank. 2005. Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- ▶ Κ. Διαμαντάρας, Ι. Μπότσης, Μηχανική Μάθηση – Α' Έκδοση, ISBN: 978-960-461-955-5, Εκδόσεις Κλειδάριθμος, 2019
- ▶ P.-N. Tan, M. Steinbach, V. Kumar, «Introduction to Data Mining», Addison Wesley, 2006
- ▶ An Introduction to Genetic Algorithms Jenna Carr
<https://www.whitman.edu/Documents/Academics/Mathematics/2014/carrjk.pdf>
- ▶ Unsupervised learning – Wei Wu https://na.uni-tuebingen.de/ex/ml_seminar_ss2022/Unsupervised_Learning%20Final.pdf

Βιβλιογραφία

<http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/>

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

<http://introtodeeplearning.com/>

<http://www.iro.umontreal.ca/~pift6266/H10/notes/deepintro.html>

https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.ht

<https://www.jeremyjordan.me/convnet-architectures/>

<https://www.analyticsvidhya.com/>

<https://www.britannica.com/topic/Deep-Blue>

<https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii-hyper-parameter-42efca01e5d7>

<https://towardsdatascience.com/understanding-and-calculating-the-number-of-parameters-in-convolution-neural-networks-cnns-fc88790d530d>

<https://www.analyticsvidhya.com/blog/2021/10/an-end-to-end-introduction-to-generative-adversarial-networksgans/>

A brief introduction to deep learning - Yangyan Li

Deep Learning, Ian Goodfellow and Yoshua Bengio and Aaron Courville

<https://www.deeplearningbook.org/>