CMP717
Image Processing

Introduction to Deep Learning

Erkut Erdem
Hacettepe University
Computer Vision Lab (HUCVL)

REVIEW

doi:10.1038/nature14539

Deep learning

Yann LeCun[1,2], Yoshua Bengio[3] & Geoffrey Hinton[4,5]

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics. Deep learning discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer. Deep convolutional nets have brought about breakthroughs in processing images, video, speech and audio, whereas recurrent nets have shone light on sequential data such as text and speech.

Machine-learning technology powers many aspects of modern society: from web searches to content filtering on social networks to recommendations on e-commerce websites, and it is increasingly present in consumer products such as cameras and smartphones. Machine-learning systems are used to identify objects in images, transcribe speech into text, match news items, posts or products with users' interests, and select relevant results of search. Increasingly, these applications make use of a class of techniques called deep learning.

Conventional machine-learning techniques were limited in their ability to process natural data in their raw form. For decades, constructing a pattern-recognition or machine-learning system required careful engineering and considerable domain expertise to design a feature extractor that transformed the raw data (such as the pixel values of an image) into a suitable internal representation or feature vector from which the learning subsystem, often a classifier, could detect or classify patterns in the input.

Representation learning is a set of methods that allows a machine to be fed with raw data and to automatically discover the representations needed for detection or classification. Deep-learning methods are representation-learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level. With the composition of enough such transformations, very complex functions can be learned. For classification tasks, higher layers of representation amplify aspects of the input that are important for discrimination and suppress irrelevant variations. An image, for example, comes in the form of an array of pixel values, and the learned features in the first layer of representation typically represent the presence or absence of edges at particular orientations and locations in the image. The second layer typically detects motifs by spotting particular arrangements of edges, regardless of small variations in the edge positions. The third layer may assemble motifs into larger combinations that correspond to parts of familiar objects, and subsequent layers would detect objects as combinations of these parts. The key aspect of deep learning is that these layers of features are not designed by human engineers: they are learned from data using a general-purpose learning procedure.

Deep learning is making major advances in solving problems that have resisted the best attempts of the artificial intelligence community for many years. It has turned out to be very good at discovering intricate structures in high-dimensional data and is therefore applicable to many domains of science, business and government. In addition to beating records in image recognition[1–4] and speech recognition[5–7], it has beaten other machine-learning techniques at predicting the activity of potential drug molecules[8], analysing particle accelerator data[9,10], reconstructing brain circuits[11], and predicting the effects of mutations in non-coding DNA on gene expression and disease[12,13]. Perhaps more surprisingly, deep learning has produced extremely promising results for various tasks in natural language understanding[14], particularly topic classification, sentiment analysis, question answering[15] and language translation[16,17].

We think that deep learning will have many more successes in the near future because it requires very little engineering by hand, so it can easily take advantage of increases in the amount of available computation and data. New learning algorithms and architectures that are currently being developed for deep neural networks will only accelerate this progress.

Supervised learning

The most common form of machine learning, deep or not, is supervised learning. Imagine that we want to build a system that can classify images as containing, say, a house, a car, a person or a pet. We first collect a large data set of images of houses, cars, people and pets, each labelled with its category. During training, the machine is shown an image and produces an output in the form of a vector of scores, one for each category. We want the desired category to have the highest score of all categories, but this is unlikely to happen before training. We compute an objective function that measures the error (or distance) between the output scores and the desired pattern of scores. The machine then modifies its internal adjustable parameters to reduce this error. These adjustable parameters, often called weights, are real numbers that can be seen as 'knobs' that define the input–output function of the machine. In a typical deep-learning system, there may be hundreds of millions of these adjustable weights, and hundreds of millions of labelled examples with which to train the machine.

To properly adjust the weight vector, the learning algorithm computes a gradient vector that, for each weight, indicates by what amount the error would increase or decrease if the weight were increased by a tiny amount. The weight vector is then adjusted in the opposite direction to the gradient vector.

The objective function, averaged over all the training examples, can

[1]Facebook AI Research, 770 Broadway, New York, New York 10003, USA. [2]New York University, 715 Broadway, New York, New York 10003, USA. [3]Department of Computer Science and Operations Research, Université de Montréal, Pavillon Andre-Aisenstadt, PO Box 6128, Centre-Ville STN Montréal, Quebec H3C 3J7, Canada. [4]Google, 1600 Amphitheatre Parkway, Mountain View, California 94043, USA. [5]Department of Computer Science, University of Toronto, 6 King's College Road, Toronto, Ontario M5S 3GA, Canada.

436 | NATURE | VOL 521 | 28 MAY 2015

What is deep learning?

Y. LeCun, Y. Bengio, G. Hinton, "**Deep Learning**", Nature, Vol. 521, 28 May 2015

"Deep learning allows computational models that are composed of **multiple processing layers to learn representations of data with multiple levels of abstraction.**"

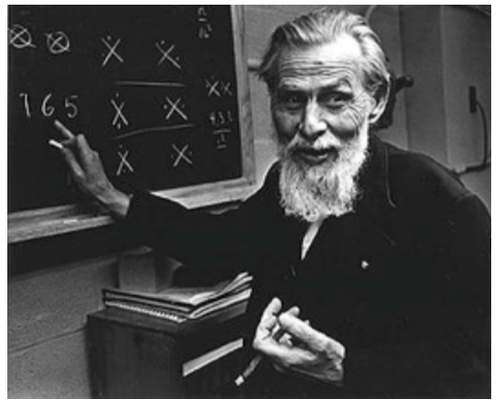
– Yann LeCun, Yoshua Bengio and Geoff Hinton

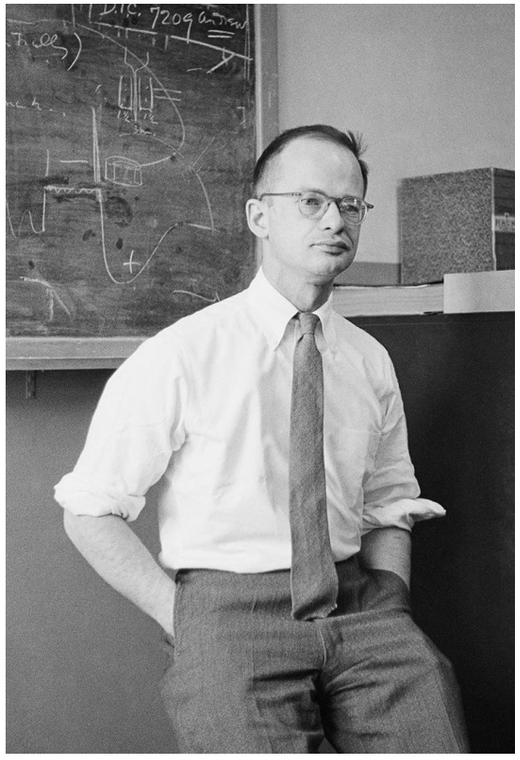
1943 – 2006:

A Prehistory of Deep Learning

1943: Warren McCulloch and Walter Pitts

- First computational model
- Neurons as logic gates (AND, OR, NOT)
- A neuron model that sums binary inputs and outputs 1 if the sum exceeds a certain threshold value, and otherwise outputs 0





Bulletin of Mathematical Biology Vol. 52, No. 1/2, pp. 99–115, 1990.
Printed in Great Britain.

0092-8240/90$3.00 + 0.00
Pergamon Press plc
Society for Mathematical Biology

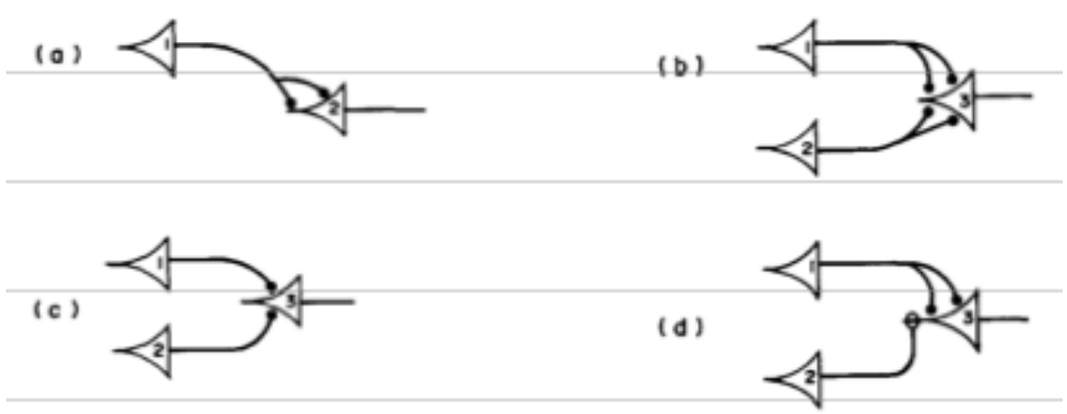
A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY*

■ WARREN S. MCCULLOCH AND WALTER PITTS
University of Illinois, College of Medicine,
Department of Psychiatry at the Illinois Neuropsychiatric Institute,
University of Chicago, Chicago, U.S.A.

Because of the "all-or-none" character of nervous activity, neural events and the relations among them can be treated by means of propositional logic. It is found that the behavior of every net can be described in these terms, with the addition of more complicated logical means for nets containing circles; and that for any logical expression satisfying certain conditions, one can find a net behaving in the fashion it describes. It is shown that many particular choices among possible neurophysiological assumptions are equivalent, in the sense that for every net behaving under one assumption, there exists another net which behaves under the other and gives the same results, although perhaps not in the same time. Various applications of the calculus are discussed.

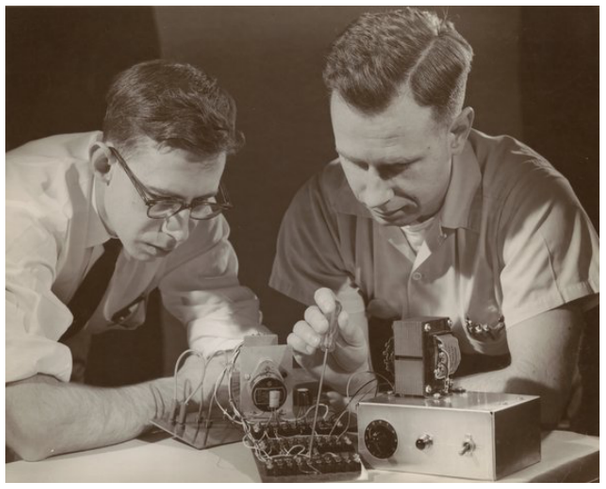
1. Introduction. Theoretical neurophysiology rests on certain cardinal assumptions. The nervous system is a net of neurons, each having a soma and an axon. Their adjunctions, or synapses, are always between the axon of one neuron and the soma of another. At any instant a neuron has some threshold, which excitation must exceed to initiate an impulse. This, except for the fact and the time of its occurence, is determined by the neuron, not by the excitation. From the point of excitation the impulse is propagated to all parts of the neuron. The velocity along the axon varies directly with its diameter, from $< 1\ ms^{-1}$ in thin axons, which are usually short, to $> 150\ ms^{-1}$ in thick axons, which are usually long. The time for axonal conduction is consequently of little importance in determining the time of arrival of impulses at points unequally remote from the same source. Excitation across synapses occurs predominantly from axonal terminations to somata. It is still a moot point whether this depends upon irreciprocity of individual synapses or merely upon prevalent anatomical configurations. To suppose the latter requires no hypothesis *ad hoc* and explains known exceptions, but any assumption as to cause is compatible with the calculus to come. No case is known in which excitation through a single synapse has elicited a nervous impulse in any neuron, whereas any neuron may be excited by impulses arriving at a sufficient number of neighboring synapses within the period of latent addition, which lasts < 0.25 ms. Observed temporal summation of impulses at greater intervals

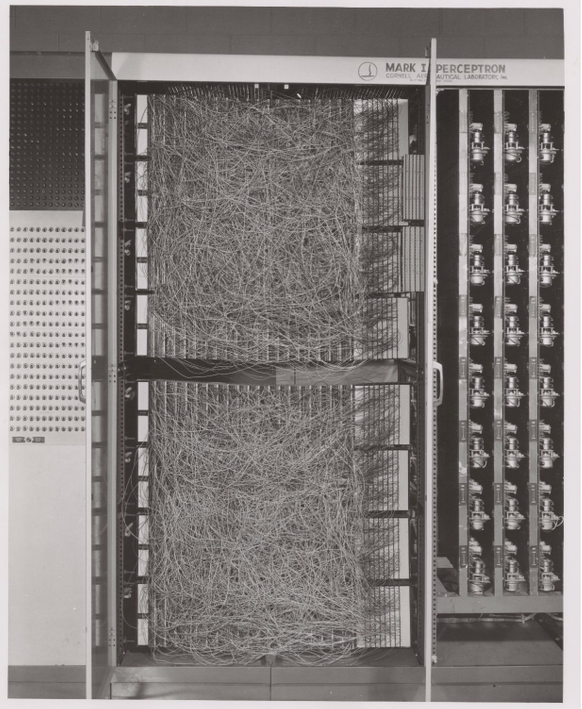
* Reprinted from the *Bulletin of Mathematical Biophysics*, Vol. 5, pp. 115–133 (1943).

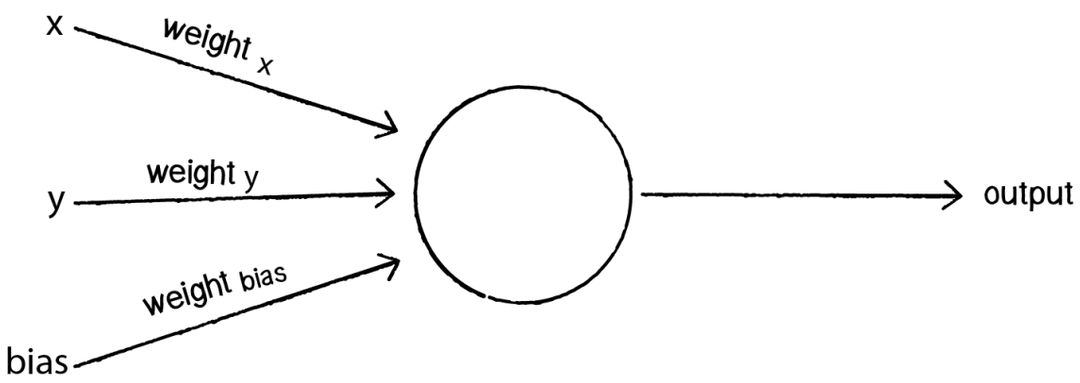


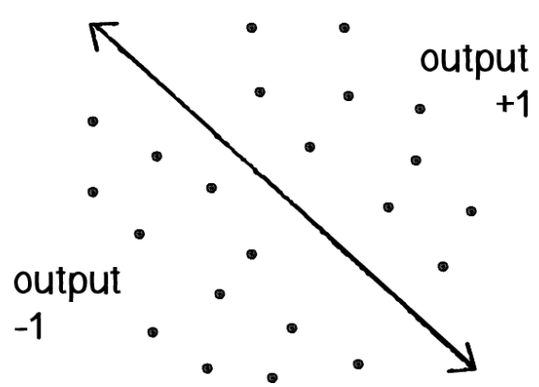
1958: Frank Rosenblatt's Perceptron

- A computational model of a **single neuron**
- Solves a **binary classification problem**
- Simple training algorithm
- Built using specialized hardware







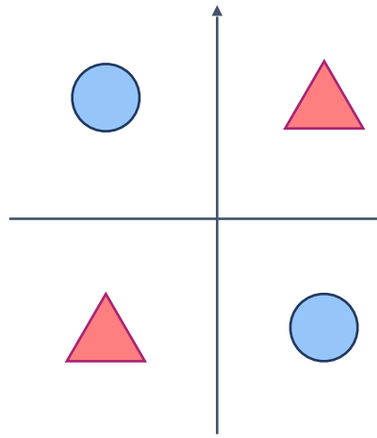


F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain", Psychological Review, Vol. 65, 1958

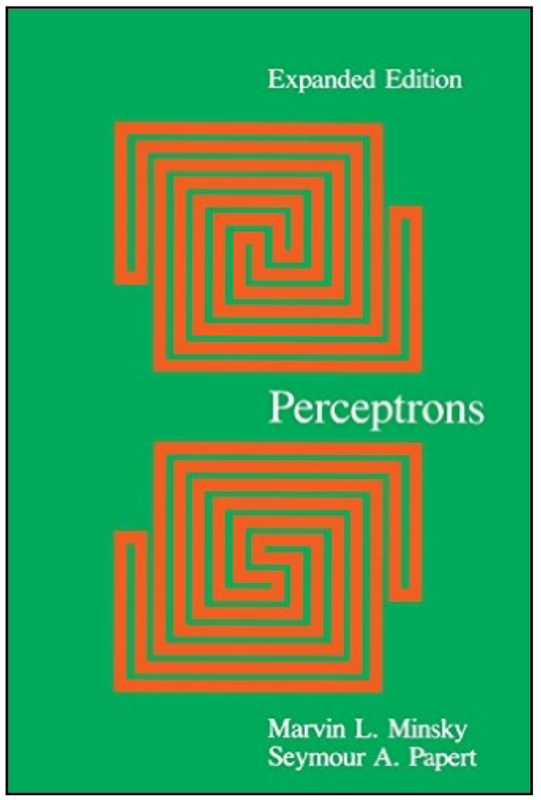
1969: Marvin Minsky and Seymour Papert

"No machine can learn to recognize X unless it possesses, at least potentially, some scheme for representing X." (p. xiii)

- Perceptrons can only represent linearly separable functions.
  - such as **XOR** Problem

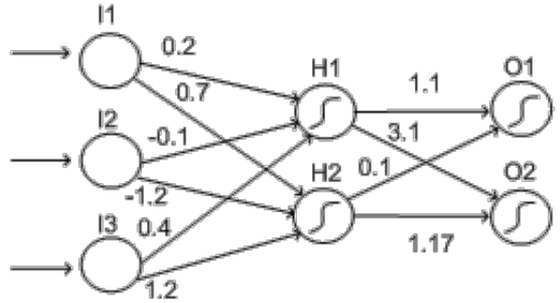


- Wrongly attributed as the reason behind the **AI winter**, a period of reduced funding and interest in AI research



1990s

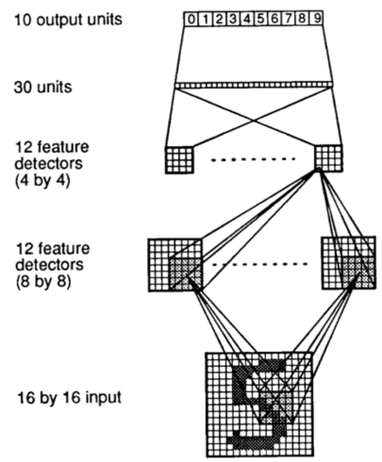
- **Multi-layer perceptrons** can theoretically learn any function (Cybenko, 1989; Hornik, 1991)



Backpropagation Through Time: What It Does and How to Do It

PAUL J. WERBOS

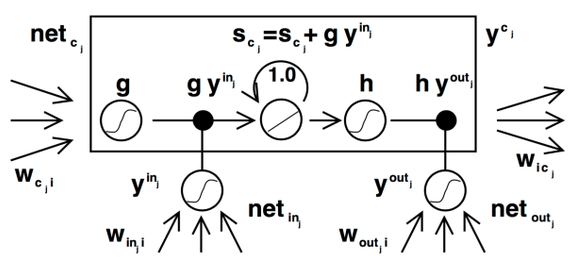
- Training multi-layer perceptrons
  - **Back-propagation** (Rumelhart, Hinton, Williams, 1986)
  - **Back-propagation through time** (BPTT) (Werbos, 1988)



Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams*

- New neural architectures
  - **Convolutional neural nets** (LeCun et al., 1989)
  - **Long-short term memory networks** (LSTM) (Schmidhuber, 1997)



Handwritten Digit Recognition with a Back-Propagation Network

Y. Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel
AT&T Bell Laboratories, Holmdel, N. J. 07733

ABSTRACT

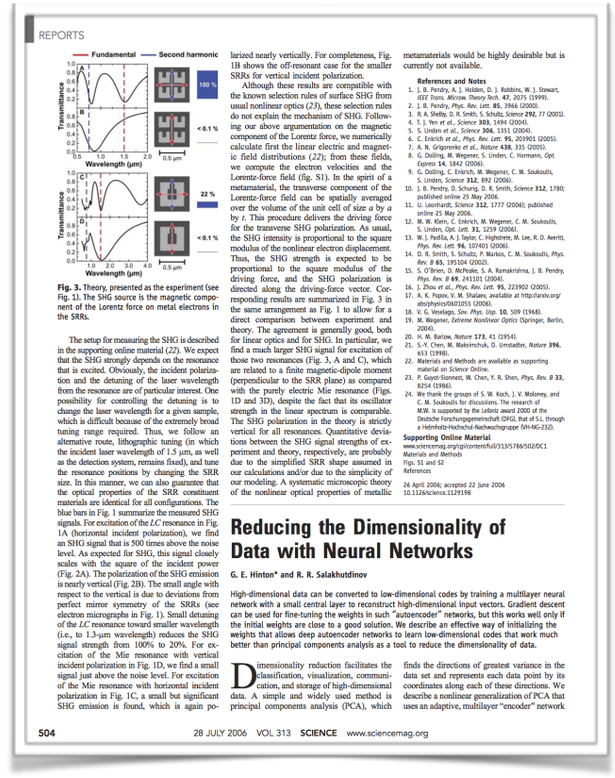
Why it failed then

- Too many parameters to learn from few labeled examples.
- “I know my features are better for this task”.
- Non-convex optimization? No, thanks.
- Black-box model, no interpretability.

- Very slow and inefficient
- Overshadowed by the success of SVMs (Cortes and Vapnik, 1995)

A major breakthrough in 2006

2006 Breakthrough: Hinton and Salakhutdinov



REPORTS

Fundamental — Second harmonic

Fig. 3. Theory, presented as the experiment (see Fig. 1). The SHG source is the magnetic component of the Lorentz force on metal electrons in the SRRs.

The setup for measuring the SHG is described in the supporting online material (*22*). We expect that the SHG strongly depends on the resonance that is excited. Obviously, the incident polarization and the detuning of the laser wavelength from the resonance are of particular interest. One possibility for controlling the detuning is to change the laser wavelength for a given sample, which is difficult because of the extremely broad tuning range required. Thus, we follow an alternative route, lithographic tuning (in which the incident laser wavelength of 1.5 μm, as well as the detection system, remains fixed), and tune the resonance positions by changing the SRR size. In this manner, we can also guarantee that the optical properties of the SRR constituent materials are identical for all configurations. The blue bars in Fig. 1 summarize the measured SHG signals. For excitation of the *LC* resonance in Fig. 1A (horizontal incident polarization), we find an SHG signal that is 500 times above the noise level. As expected for SHG, this signal closely scales with the square of the incident power (Fig. 2A). The polarization of the SHG emission is nearly vertical (Fig. 2B). The small angle with respect to the vertical is due to deviations from perfect mirror symmetry of the SRRs (see electron micrographs in Fig. 1). Small detuning of the *LC* resonance toward smaller wavelength (i.e., to 1.3-μm wavelength) reduces the SHG signal strength from 100% to 20%. For excitation of the Mie resonance with vertical incident polarization in Fig. 1D, we find a small signal just above the noise level. For excitation of the Mie resonance with horizontal incident polarization in Fig. 1C, a small but significant SHG emission is found, which is again polarized nearly vertically. For completeness, Fig. 1B shows the off-resonant case for the smaller SRRs for vertical incident polarization.

Although these results are compatible with the known selection rules of surface SHG from usual nonlinear optics (*23*), these selection rules do not explain the mechanism of SHG. Following our above argumentation on the magnetic component of the Lorentz force, we numerically calculate first the linear electric and magnetic field distributions (*22*); from these fields, we compute the electron velocities and the Lorentz-force field (fig. S1). In the spirit of a metamaterial, the transverse component of the Lorentz-force field can be spatially averaged over the volume of the unit cell of size *a* by *a* by *t*. This procedure delivers the driving force for the transverse SHG polarization. As usual, the SHG intensity is proportional to the square modulus of the nonlinear electron displacement. Thus, the SHG strength is expected to be proportional to the square modulus of the driving force, and the SHG polarization is directed along the driving-force vector. Corresponding results are summarized in Fig. 3 in the same arrangement as Fig. 1 to allow for a direct comparison between experiment and theory. The agreement is generally good, both for linear optics and for SHG. In particular, we find a much larger SHG signal for excitation of those two resonances (Fig. 3, A and C), which are related to a finite magnetic-dipole moment (perpendicular to the SRR plane) as compared with the purely electric Mie resonance (Figs. 1D and 3D), despite the fact that its oscillator strength in the linear spectrum is comparable. The SHG polarization in the theory is strictly vertical for all resonances. Quantitative deviations between the SHG signal strengths of experiment and theory, respectively, are probably due to the simplified SRR shape assumed in our calculations and/or due to the simplicity of our modeling. A systematic microscopic theory of the nonlinear optical properties of metallic metamaterials would be highly desirable but is currently not available.

References and Notes

1. J. B. Pendry, A. J. Holden, D. J. Robbins, W. J. Stewart, *IEEE Trans. Microw. Theory Tech.* **47**, 2075 (1999).
2. J. B. Pendry, *Phys. Rev. Lett.* **85**, 3966 (2000).
3. R. A. Shelby, D. R. Smith, S. Schultz, *Science* **292**, 77 (2001).
4. T. J. Yen *et al.*, *Science* **303**, 1494 (2004).
5. S. Linden *et al.*, *Science* **306**, 1351 (2004).
6. C. Enkrich *et al.*, *Phys. Rev. Lett.* **95**, 203901 (2005).
7. A. N. Grigorenko *et al.*, *Nature* **438**, 335 (2005).
8. G. Dolling, M. Wegener, S. Linden, C. Hormann, *Opt. Express* **14**, 1842 (2006).
9. G. Dolling, C. Enkrich, M. Wegener, C. M. Soukoulis, S. Linden, *Science* **312**, 892 (2006).
10. J. B. Pendry, D. Schurig, D. R. Smith, *Science* **312**, 1780; published online 25 May 2006.
11. U. Leonhardt, *Science* **312**, 1777 (2006); published online 25 May 2006.
12. M. W. Klein, C. Enkrich, M. Wegener, C. M. Soukoulis, S. Linden, *Opt. Lett.* **31**, 1259 (2006).
13. W. J. Padilla, A. J. Taylor, C. Highstrete, M. Lee, R. D. Averitt, *Phys. Rev. Lett.* **96**, 107401 (2006).
14. D. R. Smith, S. Schultz, P. Markos, C. M. Soukoulis, *Phys. Rev. B* **65**, 195104 (2002).
15. S. O'Brien, D. McPeake, S. A. Ramakrishna, J. B. Pendry, *Phys. Rev. B* **69**, 241101 (2004).
16. J. Zhou *et al.*, *Phys. Rev. Lett.* **95**, 223902 (2005).
17. A. K. Popov, V. M. Shalaev, available at http://arxiv.org/abs/physics/0601055 (2006).
18. V. G. Veselago, *Sov. Phys. Usp.* **10**, 509 (1968).
19. M. Wegener, *Extreme Nonlinear Optics* (Springer, Berlin, 2004).
20. H. M. Barlow, *Nature* **173**, 41 (1954).
21. S.-Y. Chen, M. Maksimchuk, D. Umstadter, *Nature* **396**, 653 (1998).
22. Materials and Methods are available as supporting material on *Science* Online.
23. P. Guyot-Sionnest, W. Chen, Y. R. Shen, *Phys. Rev. B* **33**, 8254 (1986).

24. We thank the groups of S. W. Koch, J. V. Moloney, and C. M. Soukoulis for discussions. The research of M.W. is supported by the Leibniz award 2000 of the Deutsche Forschungsgemeinschaft (DFG), that of S.L. through a Helmholtz-Hochschul-Nachwuchsgruppe (VH-NG-232).

Supporting Online Material

www.sciencemag.org/cgi/content/full/313/5786/502/DC1
Materials and Methods
Figs. S1 and S2
References

26 April 2006; accepted 22 June 2006
10.1126/science.1129198

Reducing the Dimensionality of Data with Neural Networks

G. E. Hinton* and R. R. Salakhutdinov

High-dimensional data can be converted to low-dimensional codes by training a multilayer neural network with a small central layer to reconstruct high-dimensional input vectors. Gradient descent can be used for fine-tuning the weights in such "autoencoder" networks, but this works well only if the initial weights are close to a good solution. We describe an effective way of initializing the weights that allows deep autoencoder networks to learn low-dimensional codes that work much better than principal components analysis as a tool to reduce the dimensionality of data.

Dimensionality reduction facilitates the classification, visualization, communication, and storage of high-dimensional data. A simple and widely used method is principal components analysis (PCA), which finds the directions of greatest variance in the data set and represents each data point by its coordinates along each of these directions. We describe a nonlinear generalization of PCA that uses an adaptive, multilayer "encoder" network

- The first solution to the **vanishing gradient problem**.
- Build the model in a layer-by-layer fashion using unsupervised learning
  - The features in early layers are already initialized or "pretrained" with some suitable features (weights).
  - Pretrained features in early layers only need to be adjusted slightly during supervised learning to achieve good results.

G. E. Hinton and R. R. Salakhutdinov, **"Reducing the dimensionality of data with neural networks"**, Science, Vol. 313, 28 July 2006.

The 2012 revolution

ImageNet Challenge

- IMAGENET Large Scale Visual Recognition Challenge (ILSVRC)
  - **1.2M** training images with **1K** categories
  - Measure top-5 classification error

Output
Scale
T-shirt
Steel drum
Drumstick
Mud turtle

✔

Output
Scale
T-shirt
Giant panda
Drumstick
Mud turtle

✗

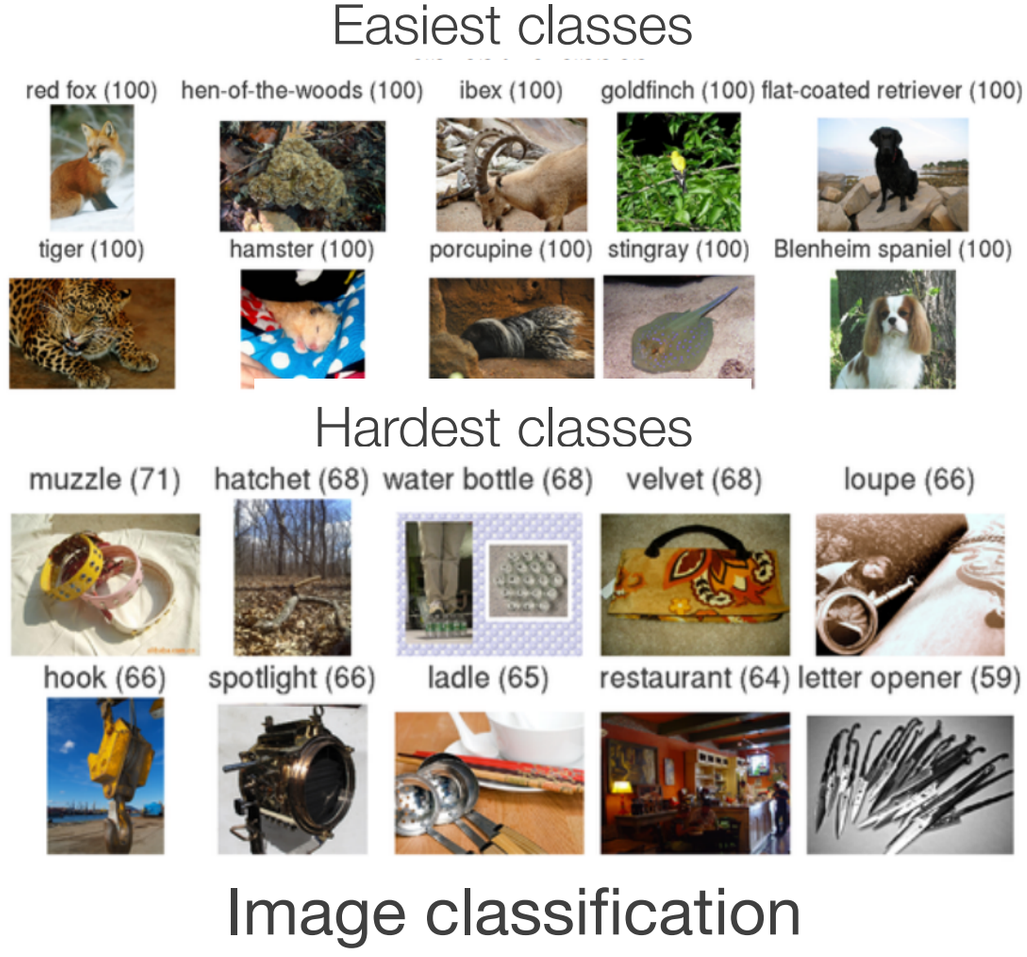


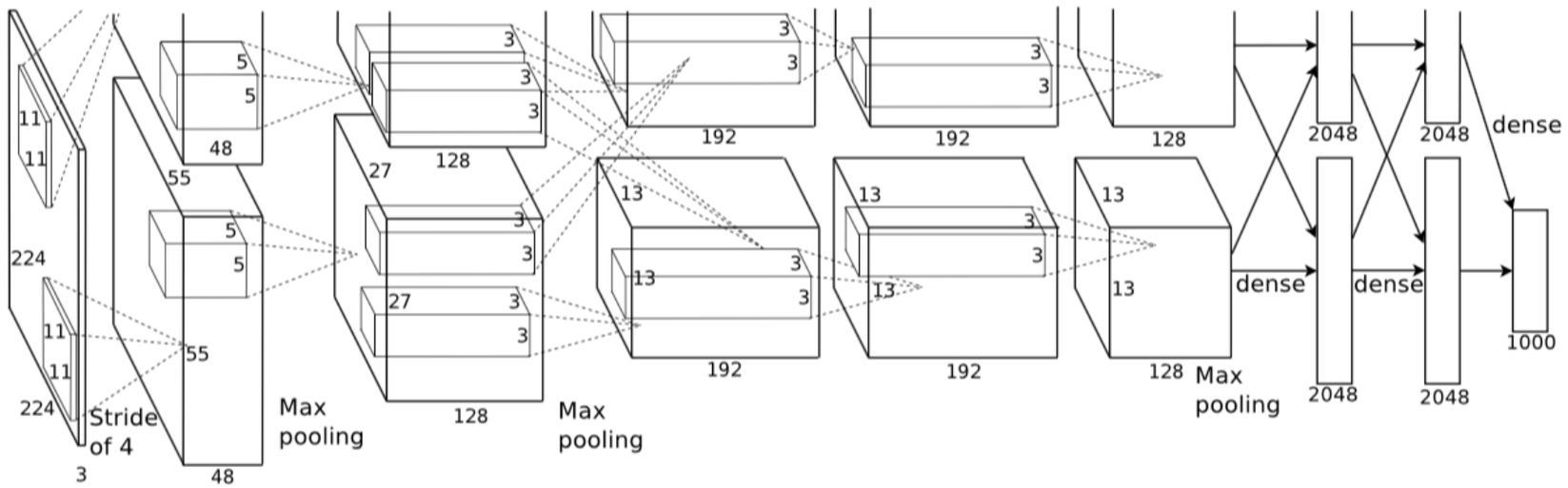
Image classification

J. Deng, Wei Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei , **"ImageNet: A Large-Scale Hierarchical Image Database"**, CVPR 2009.
O. Russakovsky et al., **"ImageNet Large Scale Visual Recognition Challenge"**, Int. J. Comput. Vis.,, Vol. 115, Issue 3, pp 211-252, 2015.

ILSVRC 2012 Competition

2012 Teams	%Error
Supervision (Toronto)	15.3
ISI (Tokyo)	26.1
VGG (Oxford)	26.9
XRCE/INRIA	27.0
UvA (Amsterdam)	29.6
INRIA/LEAR	33.4

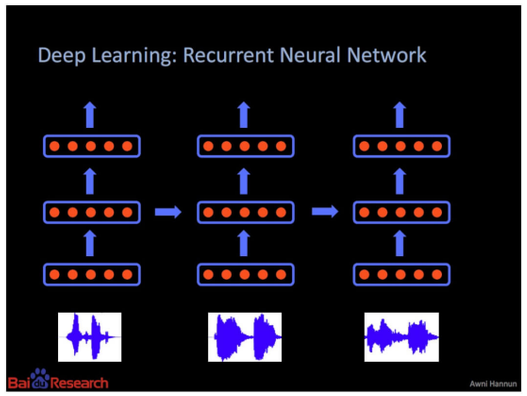
CNN based,
non-CNN based



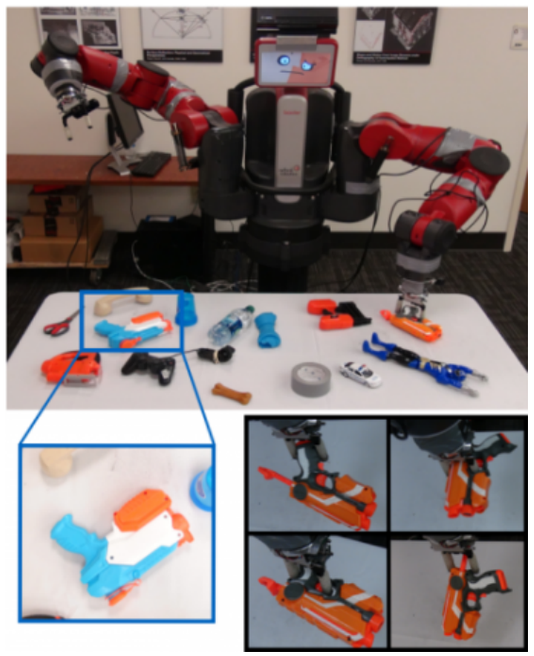
- The success of AlexNet, a deep convolutional network
  - 7 hidden layers (not counting some max pooling layers)
  - 60M parameters
- Combined several tricks
  - ReLU activation function, data augmentation, dropout

2012 – now

A Cambrian explosion in deep learning

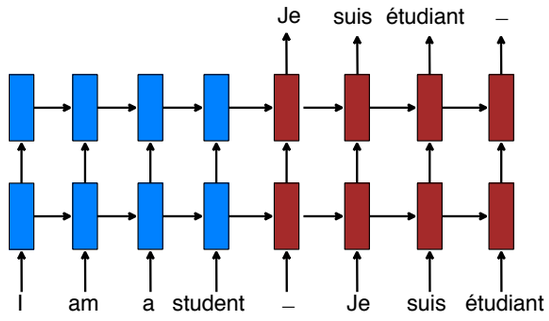


Speech recognition



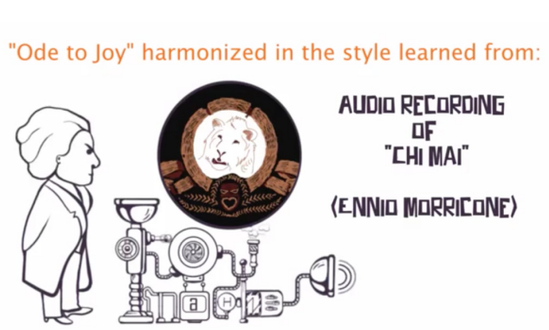
Robotics

Genomics



Machine Translation

Game Playing



Audio Generation

Self-Driving Cars

Amodei et al., **"Deep Speech 2: End-to-End Speech Recognition in English and Mandarin"**, In CoRR 2015

M.-T. Luong et al., **"Effective Approaches to Attention-based Neural Machine Translation"**, EMNLP 2015

M. Bojarski et al., **“End to End Learning for Self-Driving Cars”**, In CoRR 2016

D. Silver et al., **"Mastering the game of Go with deep neural networks and tree search"**, Nature 529, 2016

L. Pinto and A. Gupta, **“Supersizing Self-supervision: Learning to Grasp from 50K Tries and 700 Robot Hours”** ICRA 2015

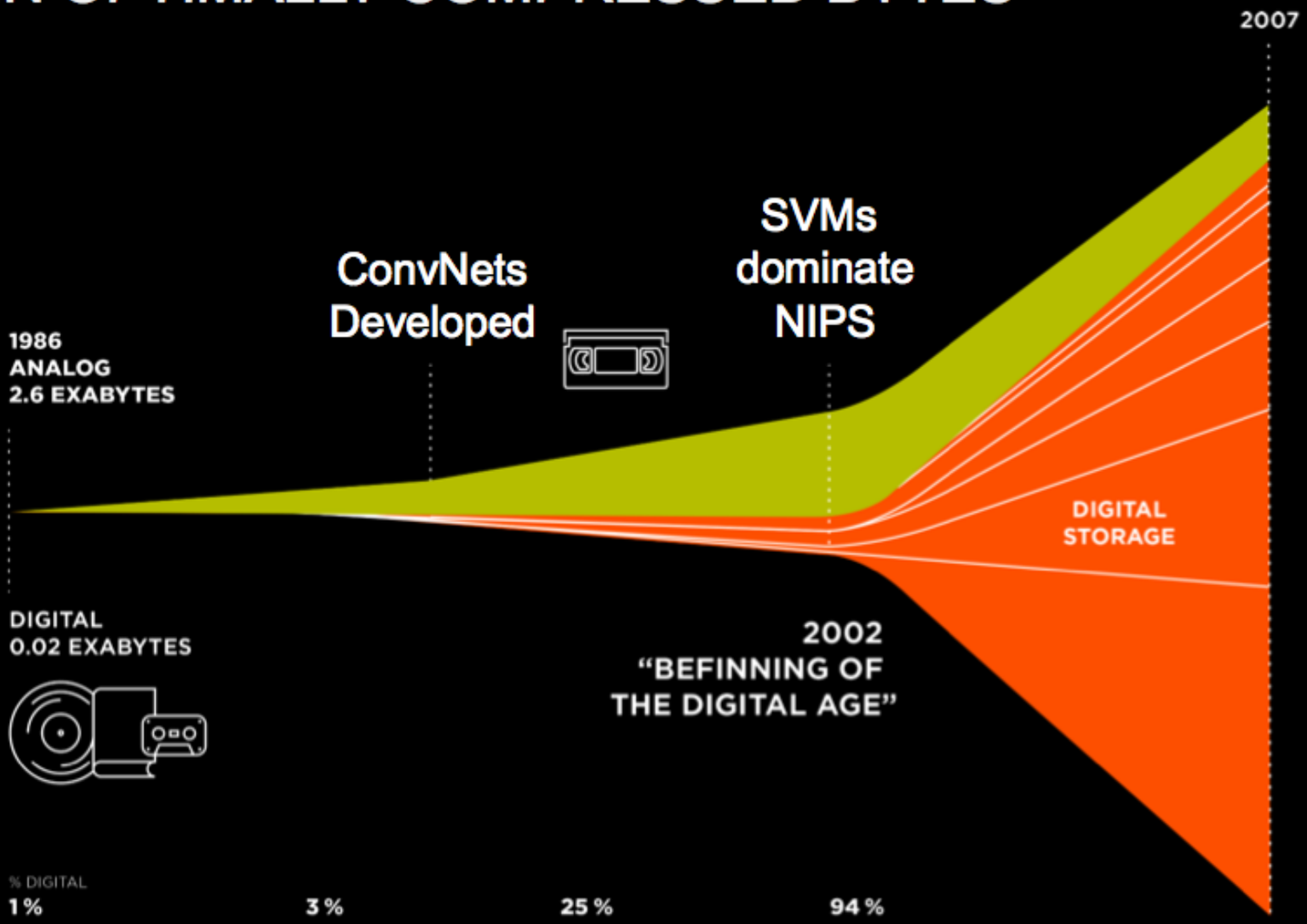
H. Y. Xiong et al., **"The human splicing code reveals new insights into the genetic determinants of disease"**, Science 347, 2015

M. Ramona et al., **"Capturing a Musician's Groove: Generation of Realistic Accompaniments from Single Song Recordings"**, In IJCAI 2015

And many more…

Why now?

GLOBAL INFORMATION STORAGE CAPACITY
IN OPTIMALLY COMPRESSED BYTES



Source: Hilbert, M., & López, P. (2011). The World's Technological Capacity to Store, Communicate, andCompute Information. Science, 332 (6025), 60-65. martinhilbert.net/worldinfocapacity.html

ANALOG
19 EXABYTES
- Paper, film, audiotape and vinyl: 6%
- Analog videotapes (VHS, etc): 94%

ANALOG ▲

DIGITAL ▼

- Portable media, flash drives: 2%
- Portable hard disks: 2.4%
- CDs & Minidisks: 6.8%
- Computer Servers and Mainframes: 8.9%
- Digital Tape: 11.8%
- DVD/Blu-Ray: 22.8%
- PC Hard Disks: 44.5%
  123 Billion Gigabytes
- Others: < 1% (incl. Chip Cards, Memory Cards, Floppy Disks, Mobile Phones, PDAs, Cameras/Camcorders, Video Games)

DIGITAL
280 EXABYTES

Slide credit: Neil Lawrence

Datasets vs. Algorithms

Year	Breakthroughs in AI	Datasets (First Available)	Algorithms (First Proposed)
1994	Human-level spontaneous speech recognition	Spoken Wall Street Journal articles and other texts (1991)	Hidden Markov Model (1984)
1997	IBM Deep Blue defeated Garry Kasparov	700,000 Grandmaster chess games, aka “The Extended Book” (1991)	Negascout planning algorithm (1983)
2005	Google’s Arabic-and Chinese-to-English translation	1.8 trillion tokens from Google Web and News pages (collected in 2005)	Statistical machine translation algorithm (1988)
2011	IBM Watson became the world Jeopardy! champion	8.6 million documents from Wikipedia, Wiktionary, and Project Gutenberg (updated in 2010)	Mixture-of-Experts (1991)
2014	Google’s GoogLeNet object classification at near-human performance	ImageNet corpus of 1.5 million labeled images and 1,000 object categories (2010)	Convolutional Neural Networks (1989)
2015	Google’s DeepMind achieved human parity in playing 29 Atari games by learning general control from video	Arcade Learning Environment dataset of over 50 Atari games (2013)	Q-learning (1992)
Average No. of Years to Breakthrough:		3 years	18 years

Table credit: Quant Quanto

Powerful Hardware

GOOGLE DATACENTER	STANFORD AI LAB
	
1,000 CPU Servers 2,000 CPUs • 16,000 cores	3 GPU-Accelerated Servers 12 GPUs • 18,432 cores
600 kWatts **$5,000,000**	**4 kWatts** **$33,000**

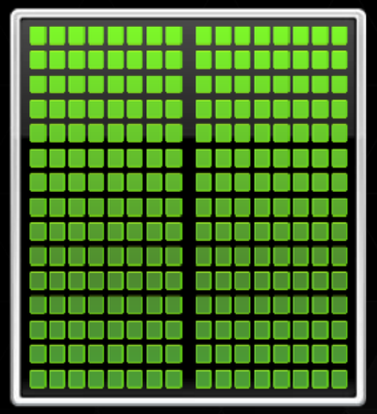
NVIDIA DGX-1

WORLD'S FIRST DEEP LEARNING SUPERCOMPUTER

170 TFLOPS FP16
8x Tesla P100 16GB
NVLink Hybrid Cube Mesh
Accelerates Major AI Frameworks
Dual Xeon
7 TB SSD Deep Learning Cache
Dual 10GbE, Quad IB 100Gb
3RU - 3200W

CPU
Optimized for
Serial Tasks

GPU Accelerator
Optimized for
Parallel Tasks



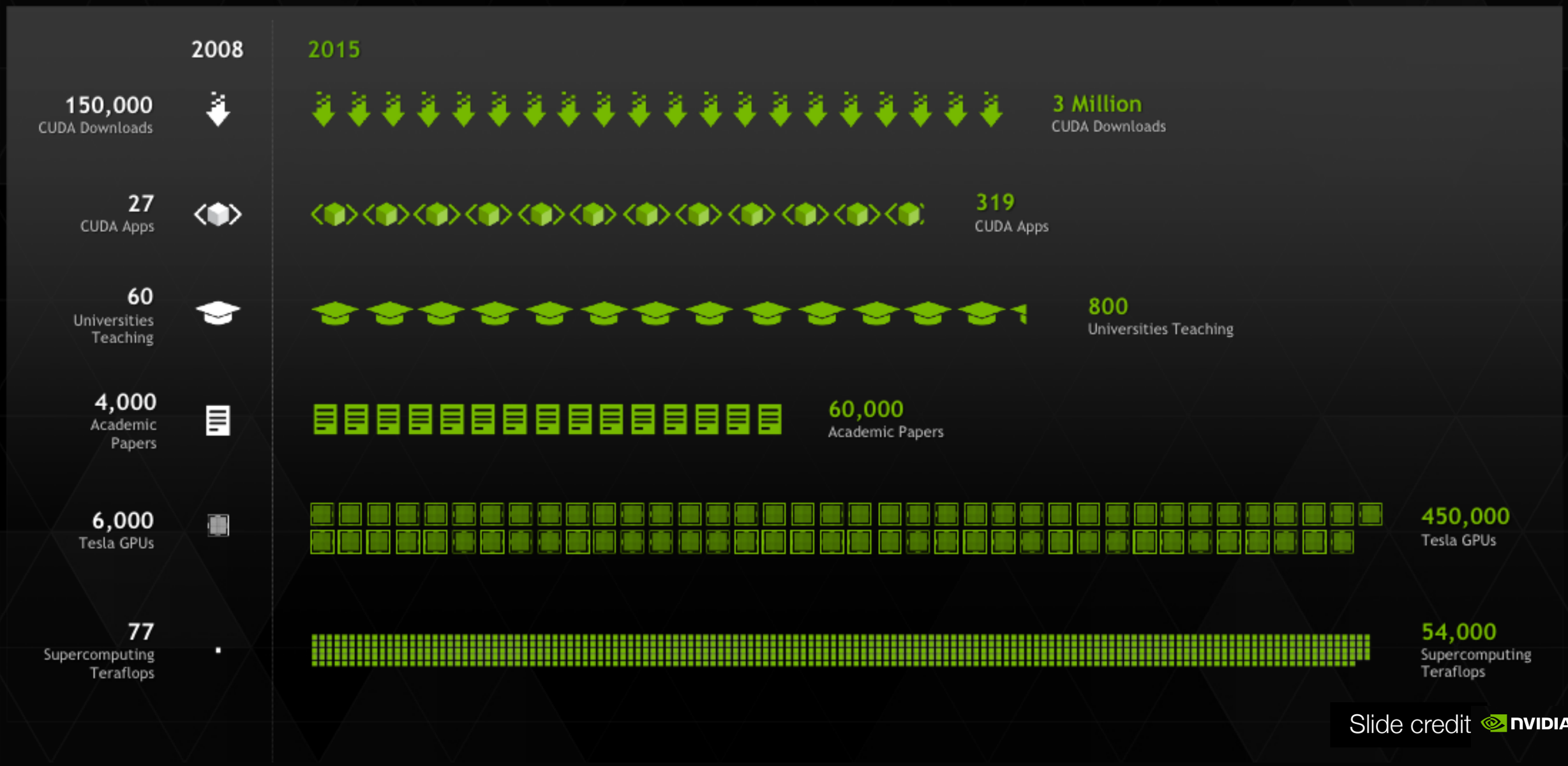
TITAN X

THE WORLD'S FASTEST GPU

8 Billion Transistors
3,072 CUDA Cores
7 TFLOPS SP / 0.2 TFLOPS DP
12GB Memory

Slide credit NVIDIA.

10X GROWTH IN GPU COMPUTING



Slide credit NVIDIA.

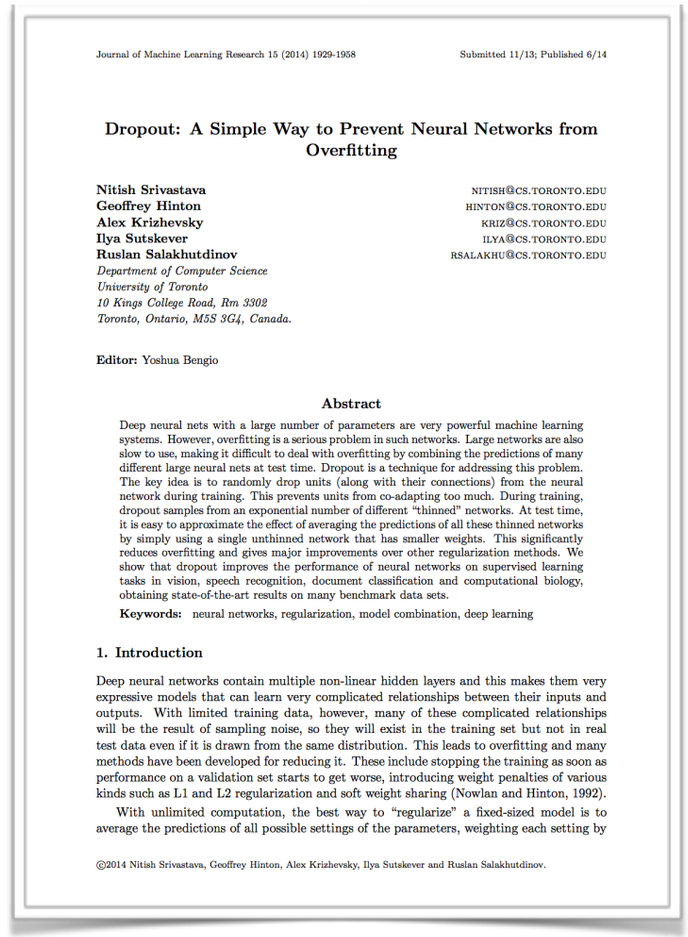
Working ideas on how to train deep architectures

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Nitish Srivastava NITISH@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU
Alex Krizhevsky KRIZ@CS.TORONTO.EDU
Ilya Sutskever ILYA@CS.TORONTO.EDU
Ruslan Salakhutdinov RSALAKHU@CS.TORONTO.EDU

Abstract

Deep neural nets with a large number of parameters are very powerful machine learning systems. However, overfitting is a serious problem in such networks. Large networks are also slow to use, making it difficult to deal with overfitting by combining the predictions of many different large neural nets at test time. Dropout is a technique for addressing this problem. The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. During training, dropout samples from an exponential number of different "thinned" networks. At test time,



Journal of Machine Learning Research 15 (2014) 1929-1958 Submitted 11/13; Published 6/14

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Nitish Srivastava NITISH@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU
Alex Krizhevsky KRIZ@CS.TORONTO.EDU
Ilya Sutskever ILYA@CS.TORONTO.EDU
Ruslan Salakhutdinov RSALAKHU@CS.TORONTO.EDU
Department of Computer Science
University of Toronto
10 Kings College Road, Rm 3302
Toronto, Ontario, M5S 3G4, Canada.

Editor: Yoshua Bengio

Abstract

Deep neural nets with a large number of parameters are very powerful machine learning systems. However, overfitting is a serious problem in such networks. Large networks are also slow to use, making it difficult to deal with overfitting by combining the predictions of many different large neural nets at test time. Dropout is a technique for addressing this problem. The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. During training, dropout samples from an exponential number of different "thinned" networks. At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single unthinned network that has smaller weights. This significantly reduces overfitting and gives major improvements over other regularization methods. We show that dropout improves the performance of neural networks on supervised learning tasks in vision, speech recognition, document classification and computational biology, obtaining state-of-the-art results on many benchmark data sets.

Keywords: neural networks, regularization, model combination, deep learning

1. Introduction

Deep neural networks contain multiple non-linear hidden layers and this makes them very expressive models that can learn very complicated relationships between their inputs and outputs. With limited training data, however, many of these complicated relationships will be the result of sampling noise, so they will exist in the training set but not in real test data even if it is drawn from the same distribution. This leads to overfitting and many methods have been developed for reducing it. These include stopping the training as soon as performance on a validation set starts to get worse, introducing weight penalties of various kinds such as L1 and L2 regularization and soft weight sharing (Nowlan and Hinton, 1992).

With unlimited computation, the best way to "regularize" a fixed-sized model is to average the predictions of all possible settings of the parameters, weighting each setting by

- Better Learning Regularization (e.g. **Dropout**)

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, **"Dropout: A Simple Way to Prevent Neural Networks from Overfitting"**,
JMLR Vol. 15, No. 1,

Working ideas on how to train deep architectures

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Sergey Ioffe
Google Inc., *sioffe@google.com*

Christian Szegedy
Google Inc., *szegedy@google.com*

Abstract

Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. We refer to this phenomenon as *internal covariate shift*, and address the problem by normalizing layer inputs. Our method draws its strength from making normalization a part of the model architecture and performing the normalization *for each training mini-batch*. Batch Normalization allows us to use much higher learning rates and be less careful about initialization. It also acts as a regularizer, in some cases eliminating the need for Dropout. Applied to a state-of-the-art image classification model, Batch Normalization achieves the same accuracy with 14 times fewer training steps, and beats the original model by a significant margin. Using an ensemble of batch-normalized networks, we improve upon the best published result on ImageNet classification: reaching 4.9% top-5 validation error (and 4.8% test error), exceeding the accuracy of human raters.

Using mini-batches of examples, as opposed to one example at a time, is helpful in several ways. First, the gradient of the loss over a mini-batch is an estimate of the gradient over the training set, whose quality improves as the batch size increases. Second, computation over a batch can be much more efficient than m computations for individual examples, due to the parallelism afforded by the modern computing platforms.

While stochastic gradient is simple and effective, it requires careful tuning of the model hyper-parameters, specifically the learning rate used in optimization, as well as the initial values for the model parameters. The training is complicated by the fact that the inputs to each layer


Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Sergey Ioffe
Google Inc., *sioffe@google.com*

Christian Szegedy
Google Inc., *szegedy@google.com*

Abstract

Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. We refer to this phenomenon as *internal covariate shift*, and address the problem by normalizing layer inputs. Our method draws its strength from making normalization a part of the model architecture and performing the normalization *for each training mini-batch*. Batch Normalization allows us to use much higher learning rates and be less careful about initialization. It also acts as a regularizer, in some cases eliminating the need for Dropout. Applied to a state-of-the-art image classification model, Batch Normalization achieves the same accuracy with 14 times fewer training steps, and beats the original model by a significant margin. Using an ensemble of batch-normalized networks, we improve upon the best published result on ImageNet classification: reaching 4.9% top-5 validation error (and 4.8% test error), exceeding the accuracy of human raters.

1 Introduction

Deep learning has dramatically advanced the state of the art in vision, speech, and many other areas. Stochastic gradient descent (SGD) has proved to be an effective way of training deep networks, and SGD variants such as momentum (Sutskever et al., 2013) and Adagrad (Duchi et al., 2011) have been used to achieve state of the art performance. SGD optimizes the parameters Θ of the network, so as to minimize the loss

$$\Theta = \arg\min_{\Theta} \frac{1}{N} \sum_{i=1}^{N} \ell(\mathbf{x}_i, \Theta)$$

where $\mathbf{x}_{1\ldots N}$ is the training data set. With SGD, the training proceeds in steps, and at each step we consider a *mini-batch* $\mathbf{x}_{1\ldots m}$ of size m. The mini-batch is used to approximate the gradient of the loss function with respect to the parameters, by computing

$$\frac{1}{m} \frac{\partial \ell(\mathbf{x}_i, \Theta)}{\partial \Theta}.$$

Using mini-batches of examples, as opposed to one example at a time, is helpful in several ways. First, the gradient of the loss over a mini-batch is an estimate of the gradient over the training set, whose quality improves as the batch size increases. Second, computation over a batch can be much more efficient than m computations for individual examples, due to the parallelism afforded by the modern computing platforms.

While stochastic gradient is simple and effective, it requires careful tuning of the model hyper-parameters, specifically the learning rate used in optimization, as well as the initial values for the model parameters. The training is complicated by the fact that the inputs to each layer are affected by the parameters of all preceding layers – so that small changes to the network parameters amplify as the network becomes deeper.

The change in the distributions of layers' inputs presents a problem because the layers need to continuously adapt to the new distribution. When the input distribution to a learning system changes, it is said to experience *covariate shift* (Shimodaira, 2000). This is typically handled via domain adaptation (Jiang, 2008). However, the notion of covariate shift can be extended beyond the learning system as a whole, to apply to its parts, such as a sub-network or a layer. Consider a network computing

$$\ell = F_2(F_1(\mathrm{u}, \Theta_1), \Theta_2)$$

where F_1 and F_2 are arbitrary transformations, and the parameters Θ_1, Θ_2 are to be learned so as to minimize the loss ℓ. Learning Θ_2 can be viewed as if the inputs $\mathrm{x} = F_1(\mathrm{u}, \Theta_1)$ are fed into the sub-network

$$\ell = F_2(\mathrm{x}, \Theta_2).$$

For example, a gradient descent step

$$\Theta_2 \leftarrow \Theta_2 - \frac{\alpha}{m} \sum_{i=1}^{m} \frac{\partial F_2(\mathbf{x}_i, \Theta_2)}{\partial \Theta_2}$$

(for batch size m and learning rate α) is exactly equivalent to that for a stand-alone network F_2 with input x. Therefore, the input distribution properties that make training more efficient – such as having the same distribution between the training and test data – apply to training the sub-network as well. As such it is advantageous for the distribution of x to remain fixed over time. Then, Θ_2 does

1

- Better Optimization Conditioning (e.g. **Batch Normalization**)

S. Ioffe, C. Szegedy, **"Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift"**, In ICML 2015

Working ideas on how to train deep architectures

Deep Residual Learning for Image Recognition

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

Abstract

Deeper neural networks are more difficult to train. We present a residual learning framework to ease the training of networks that are substantially deeper than those used previously. We explicitly reformulate the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. We provide comprehensive empirical evidence showing that these residual networks are easier to optimize, and can gain accuracy from considerably increased depth. On the ImageNet dataset we evaluate residual nets with a depth of up to 152 layers—8× deeper than VGG nets [41] but still having lower complexity. An ensemble of these residual nets achieves 3.57% error

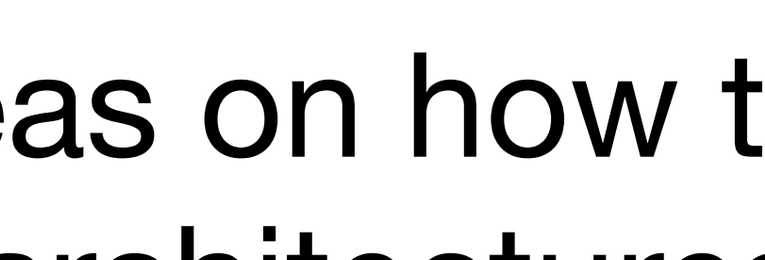
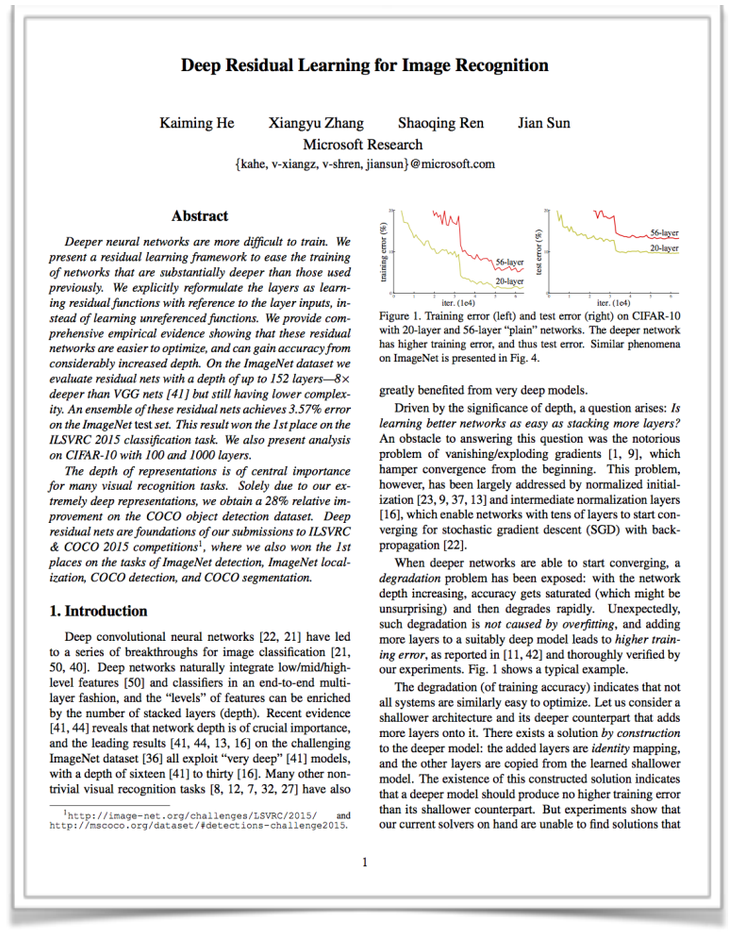


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

greatly benefited from very deep models.

Driven by the significance of depth, a question arises: *Is*

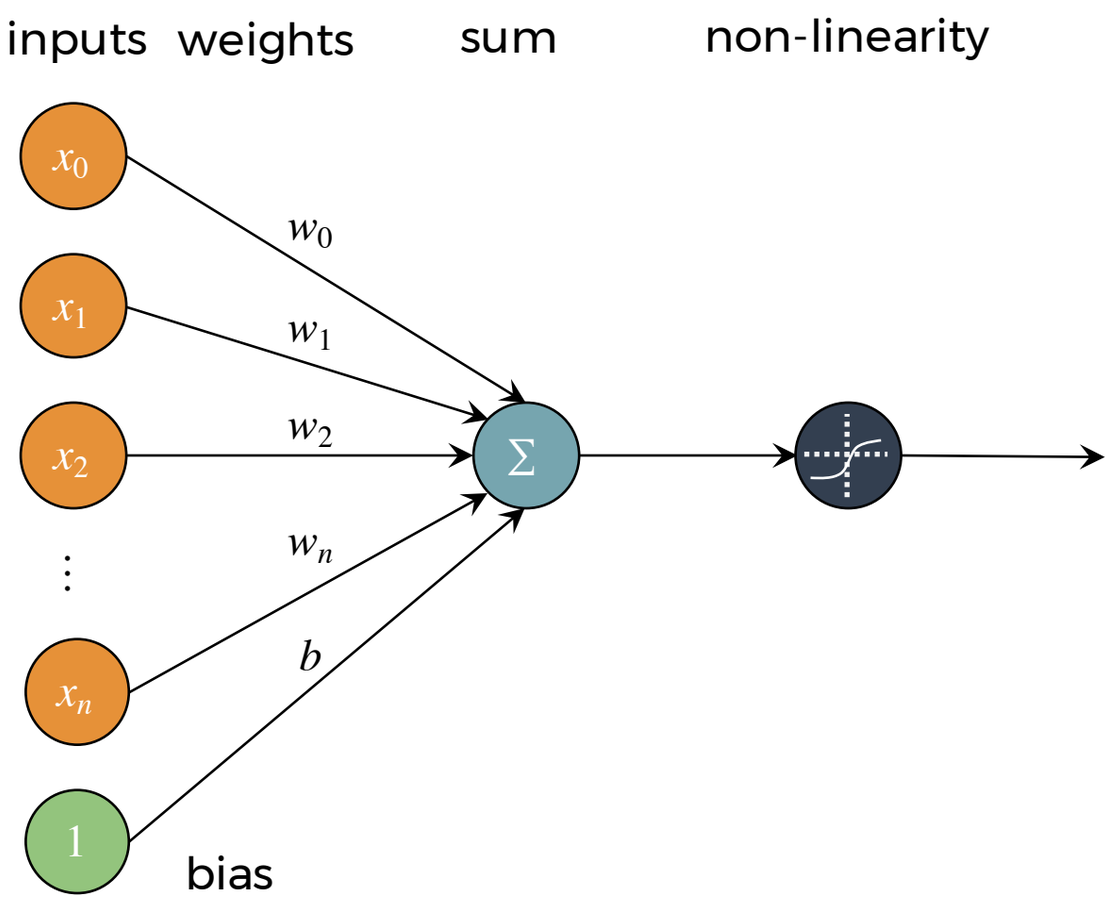


- Better neural achitectures (e.g. **Residual Nets**)

K. He, X. Zhang, S. Ren, J. Sun, **"Deep Residual Learning for Image Recognition"**, In CVPR 2016

Let’s make a review of neural networks

The Perceptron



Perceptron Forward Pass

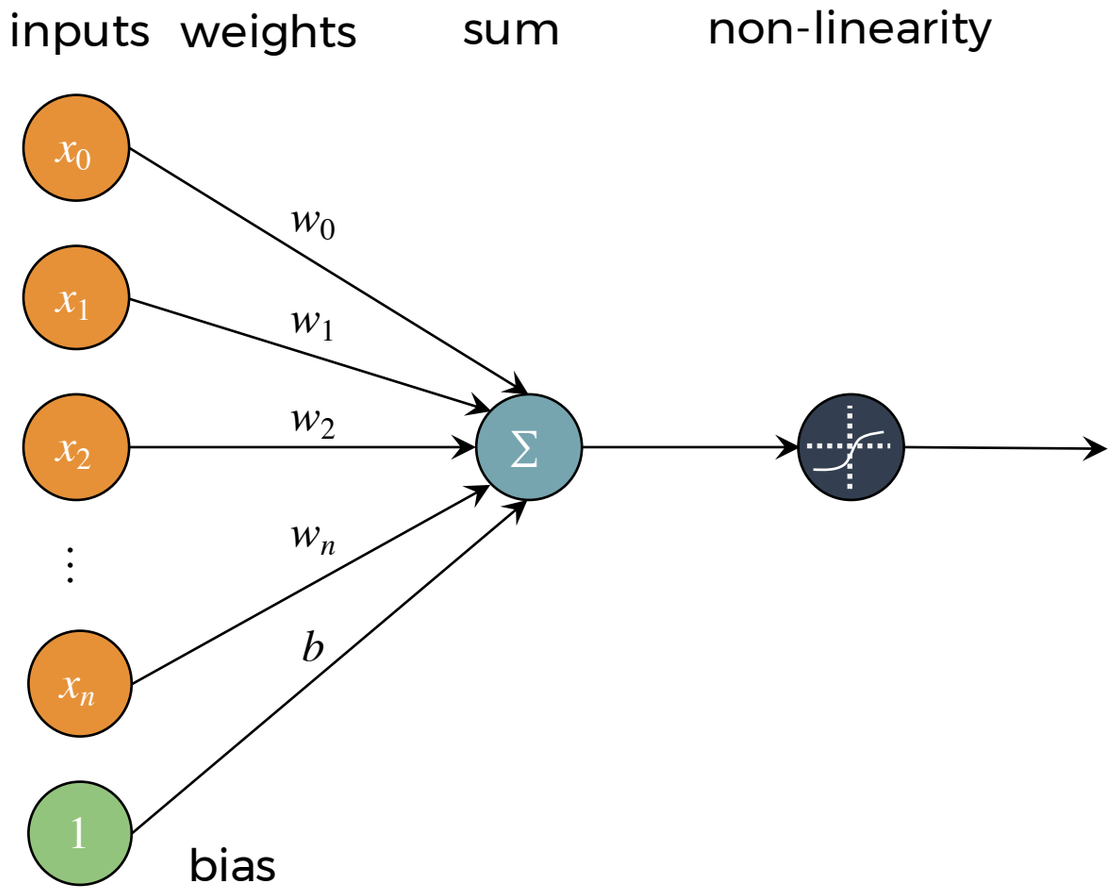
- Neuron pre-activation (or input activation)

$$a(\mathbf{x}) = b + \sum_i w_i x_i = b + \mathbf{w}^\top \mathbf{x}$$

- Neuron output activation:

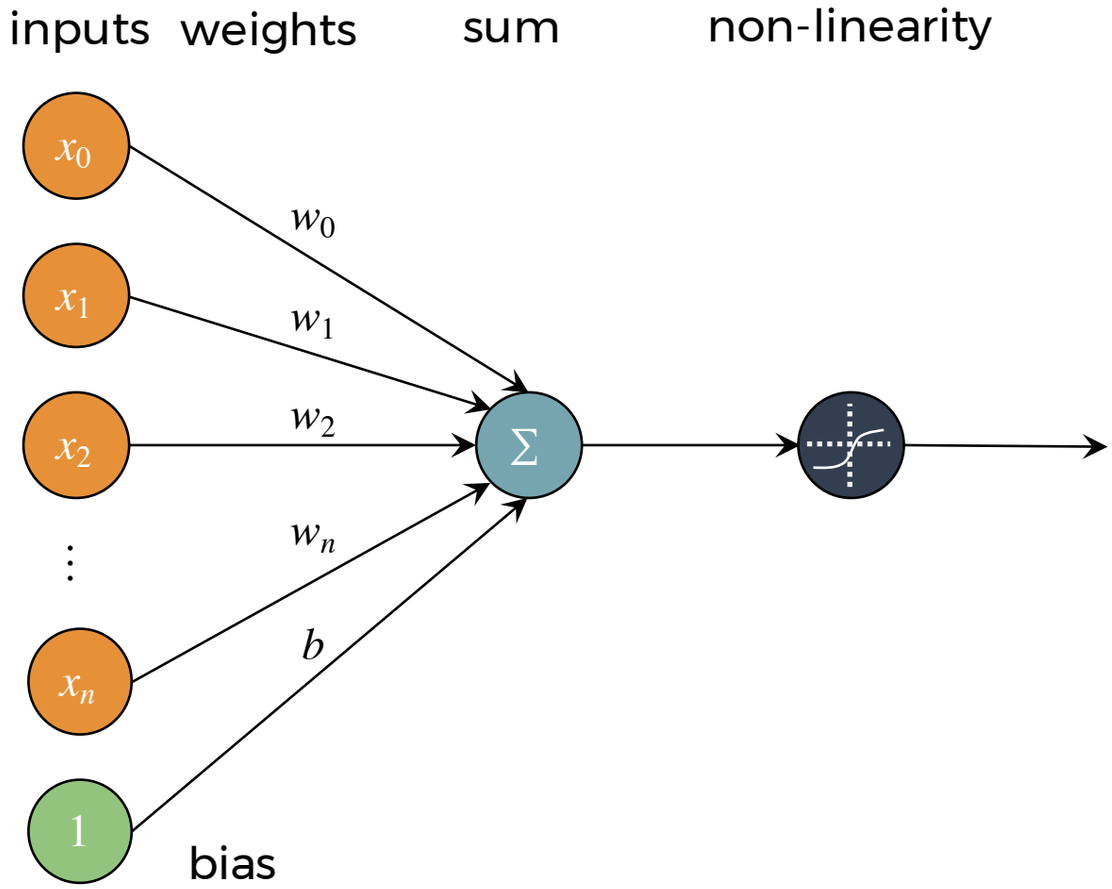
$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$

where
  w are the weights (parameters)
  b is the bias term
  g(·) is called the activation function



Output Activation of The Neuron

$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$



Range is determined by $g(\cdot)$

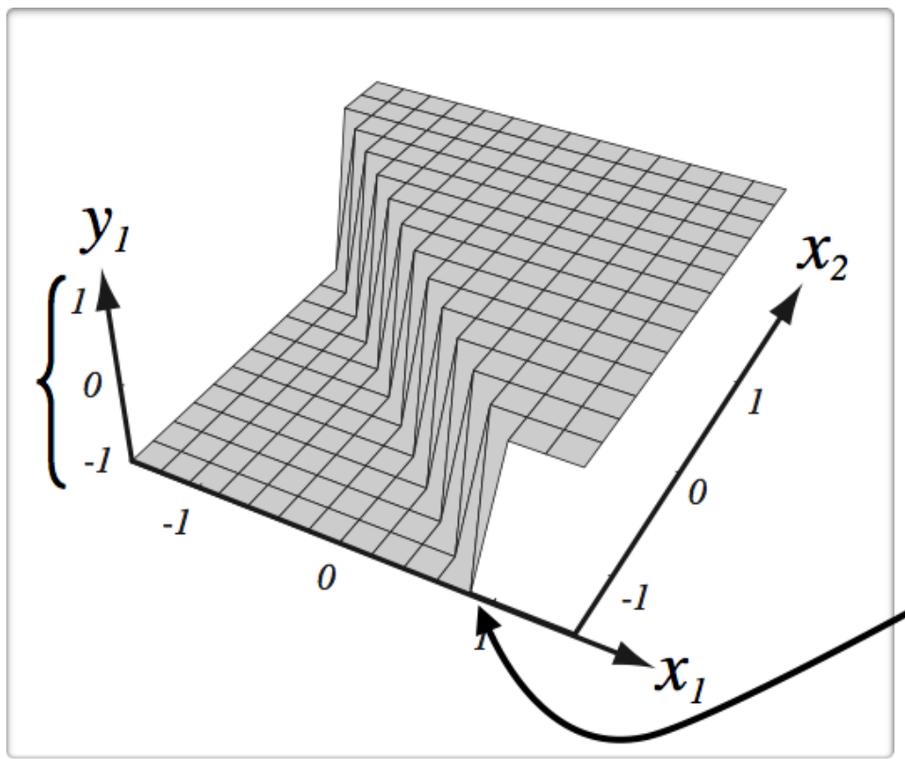


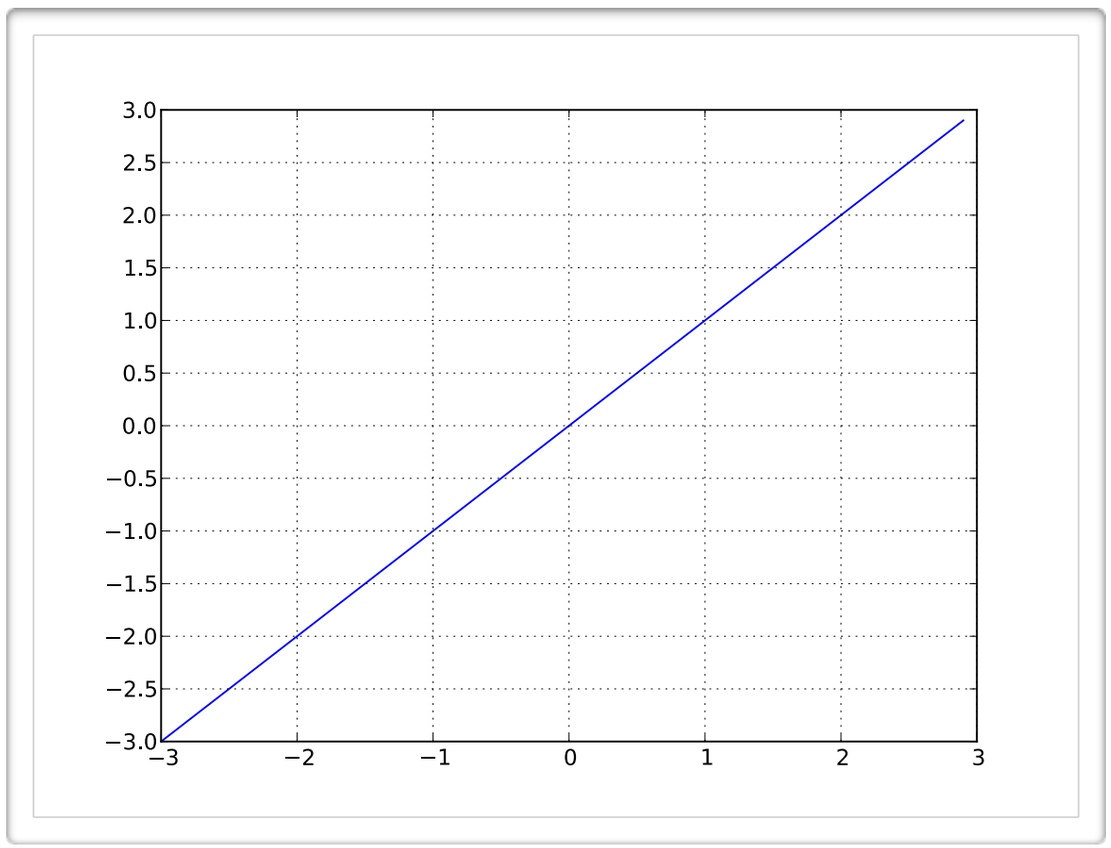
Image credit: Pascal Vincent

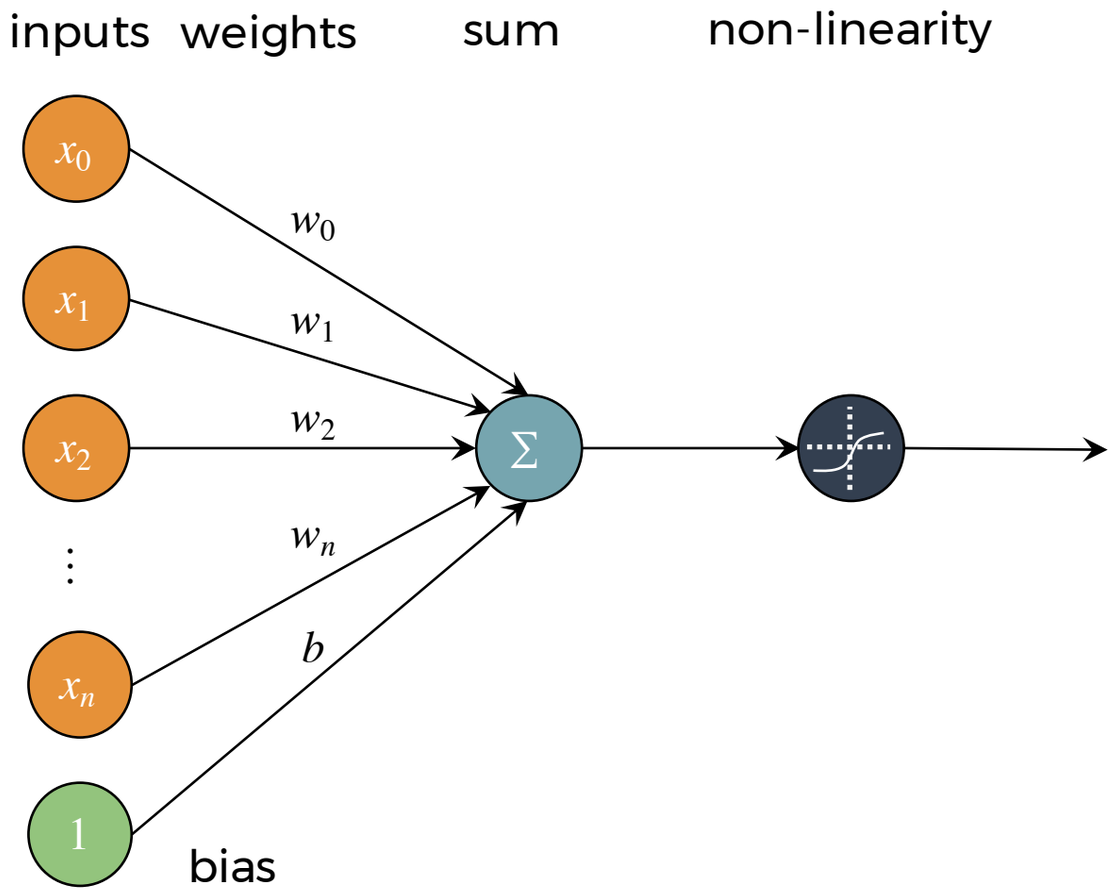
Bias only changes the position of the riff

Linear Activation Function

$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$

$$g(a) = a$$



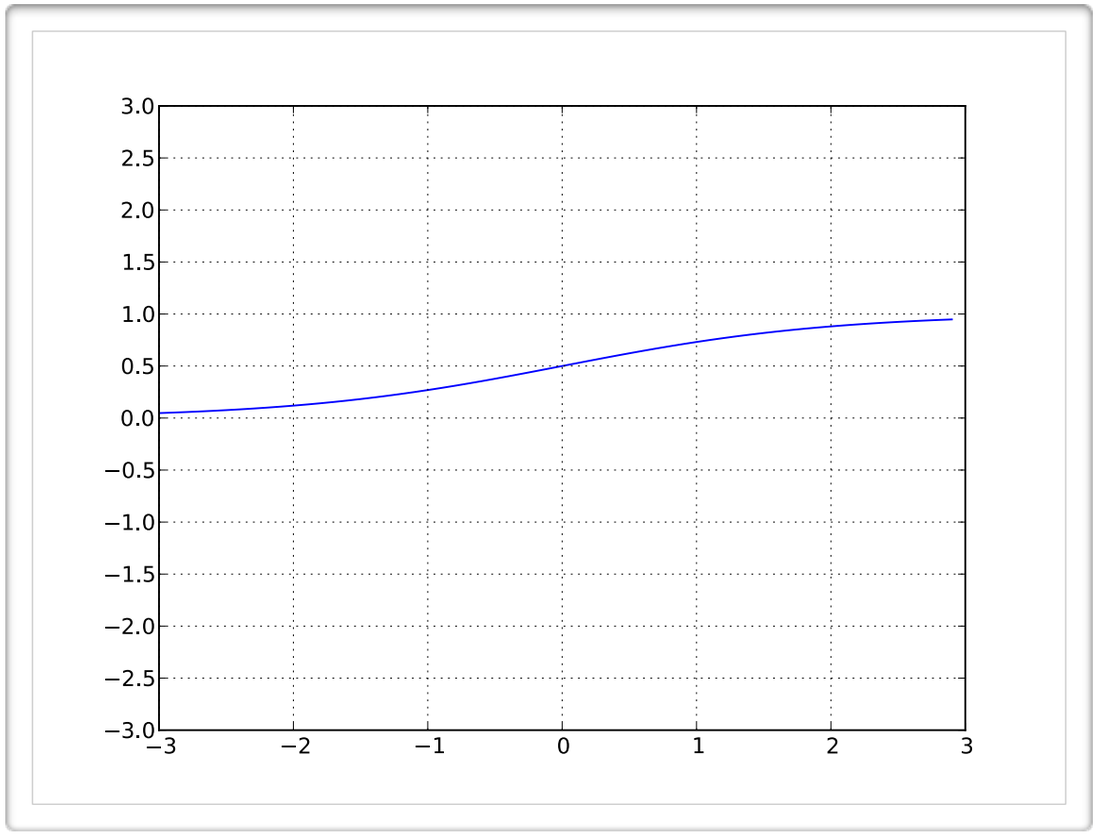


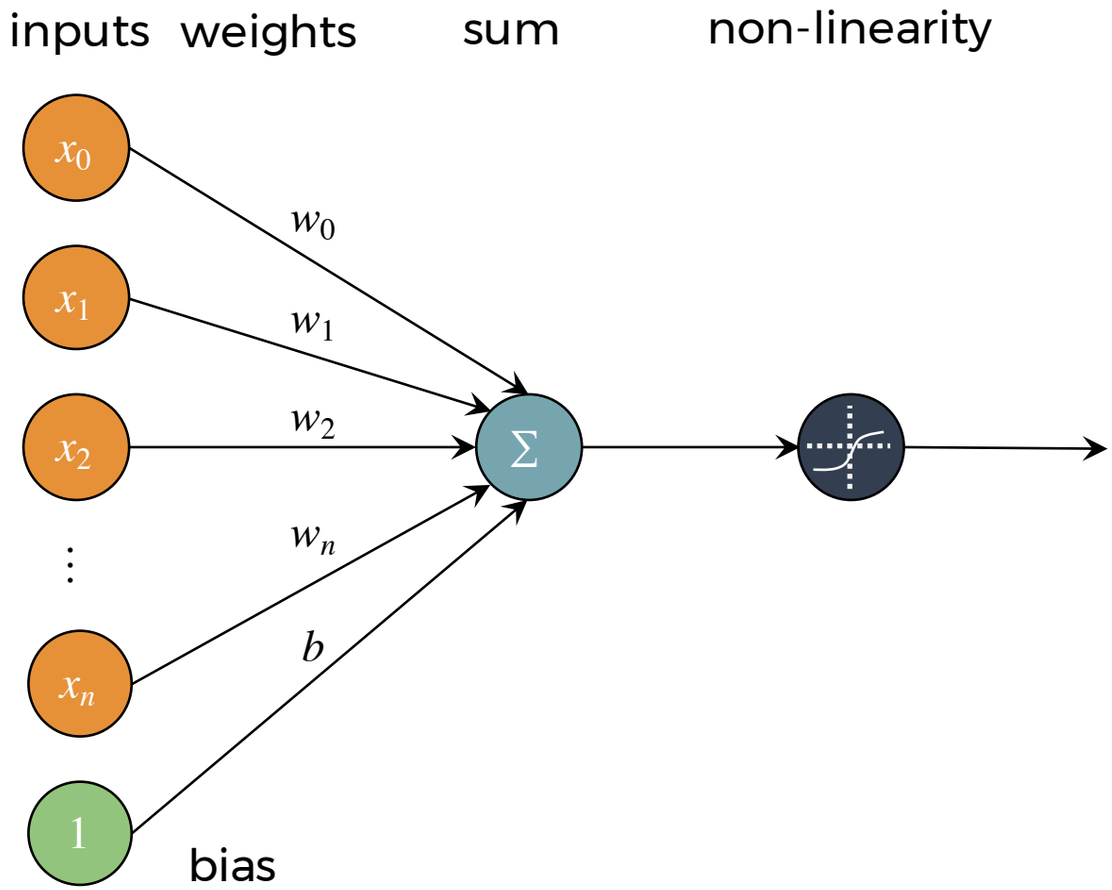
No nonlinear transformation
No input squashing

Sigmoid Activation Function

$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$

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





Squashes the neuron's output between 0 and 1

Always positive

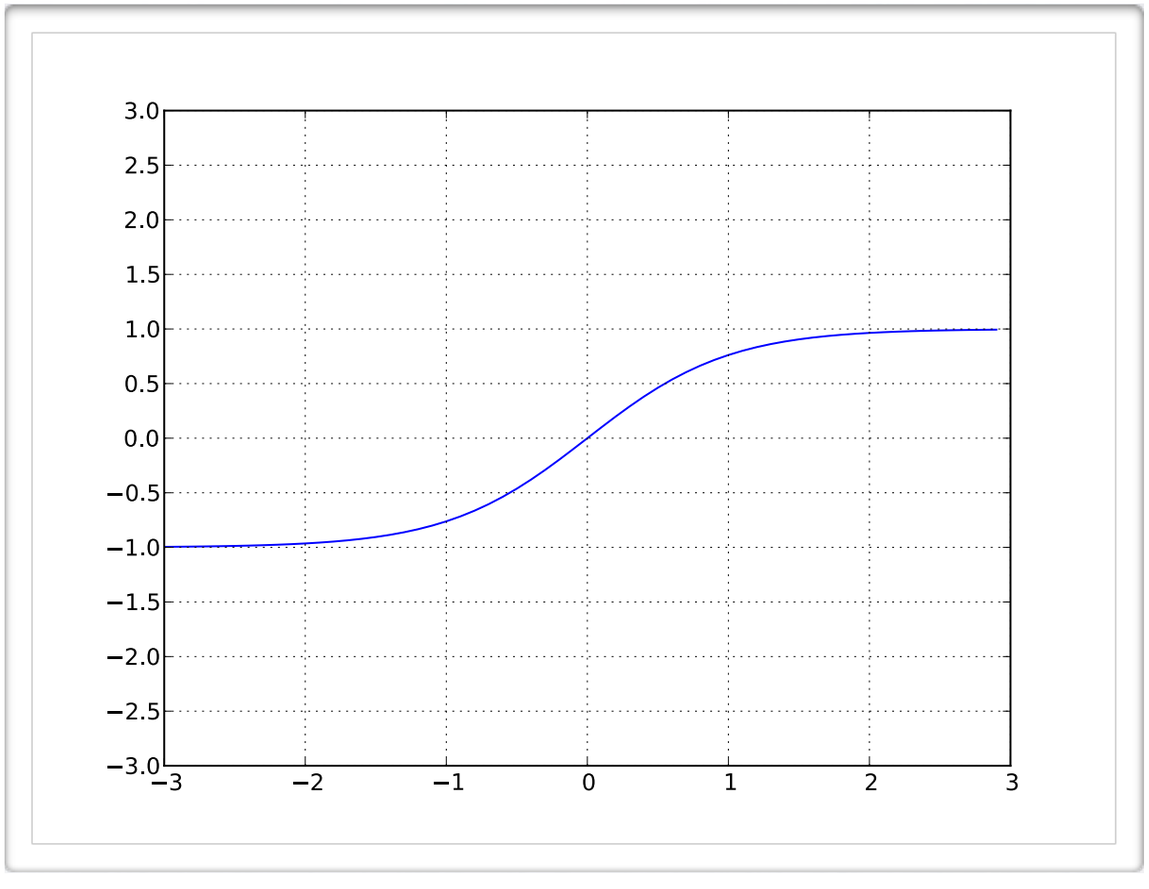
Bounded

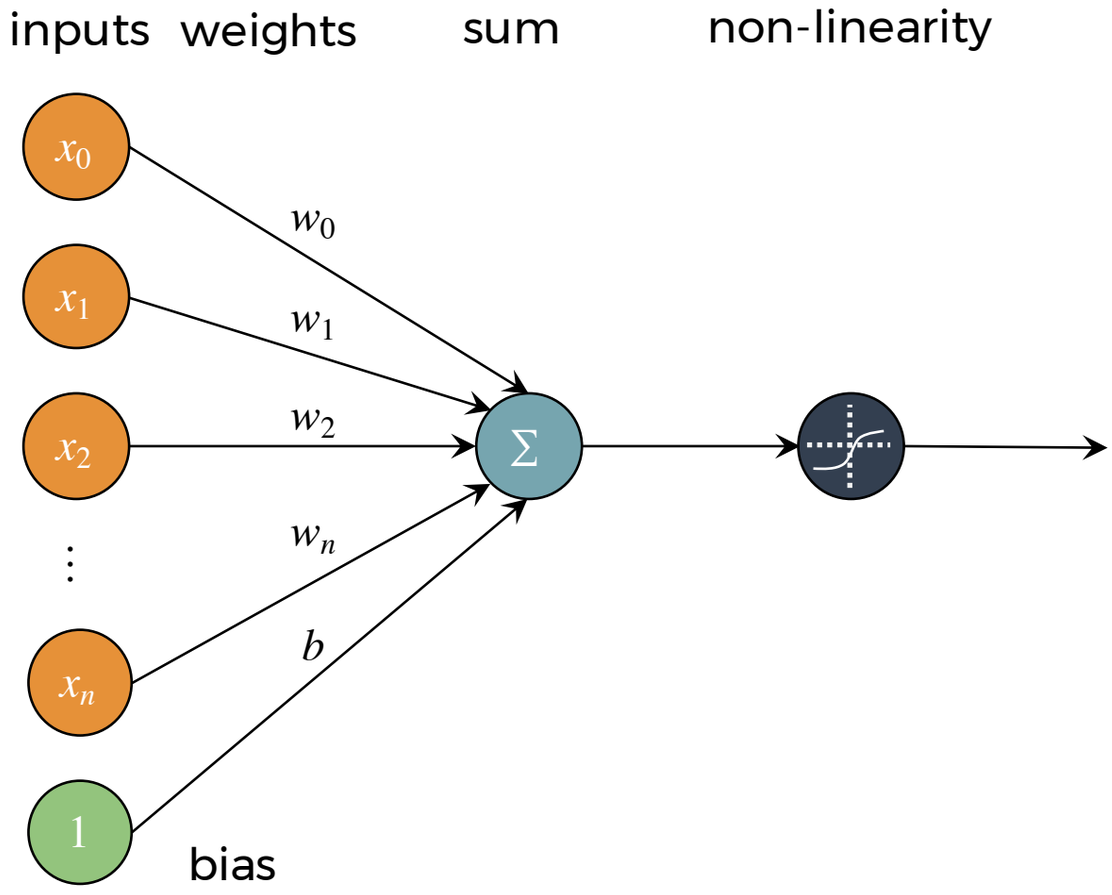
Strictly Increasing

Hyperbolic Tangent (tanh) Activation Function

$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$

$$g(a) = \tanh(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)} = \frac{\exp(2a) - 1}{\exp(2a) + 1}$$





Squashes the neuron's output between -1 and 1

Can be positive or negative

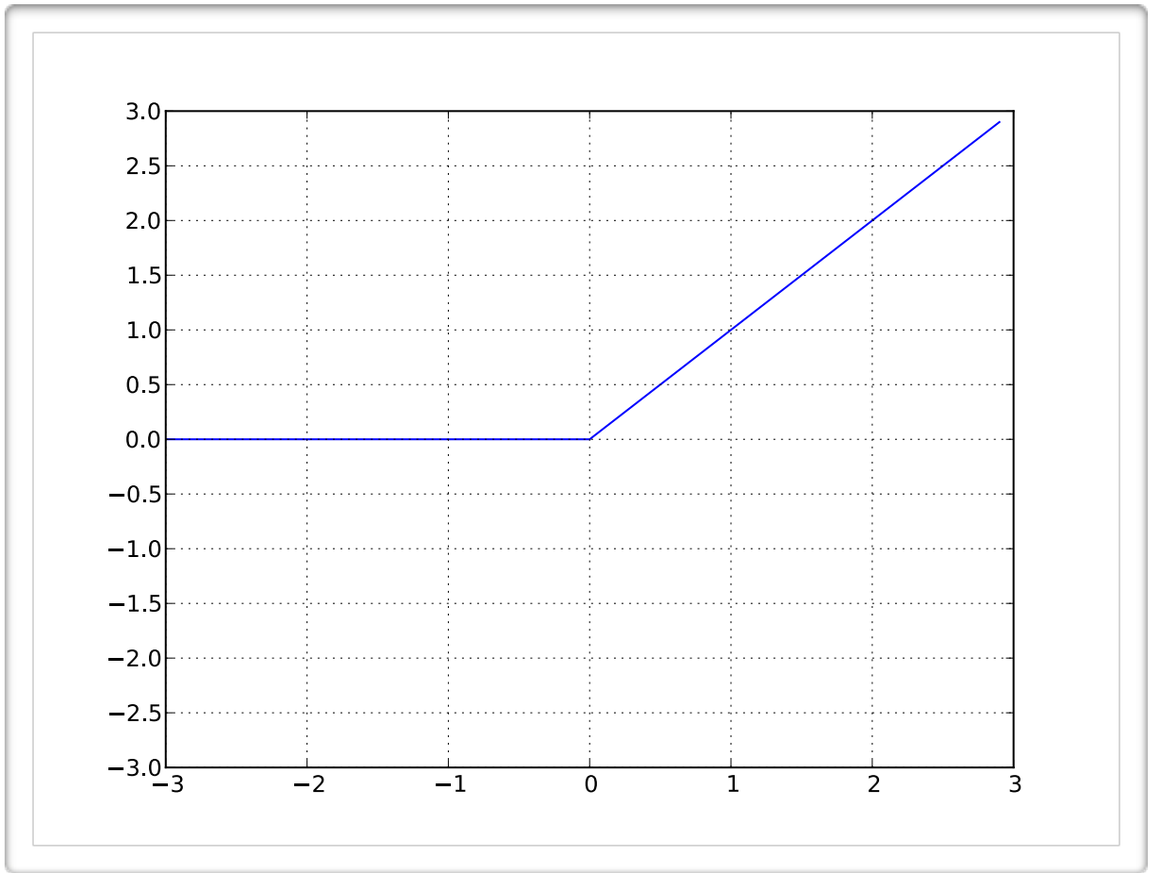
Bounded

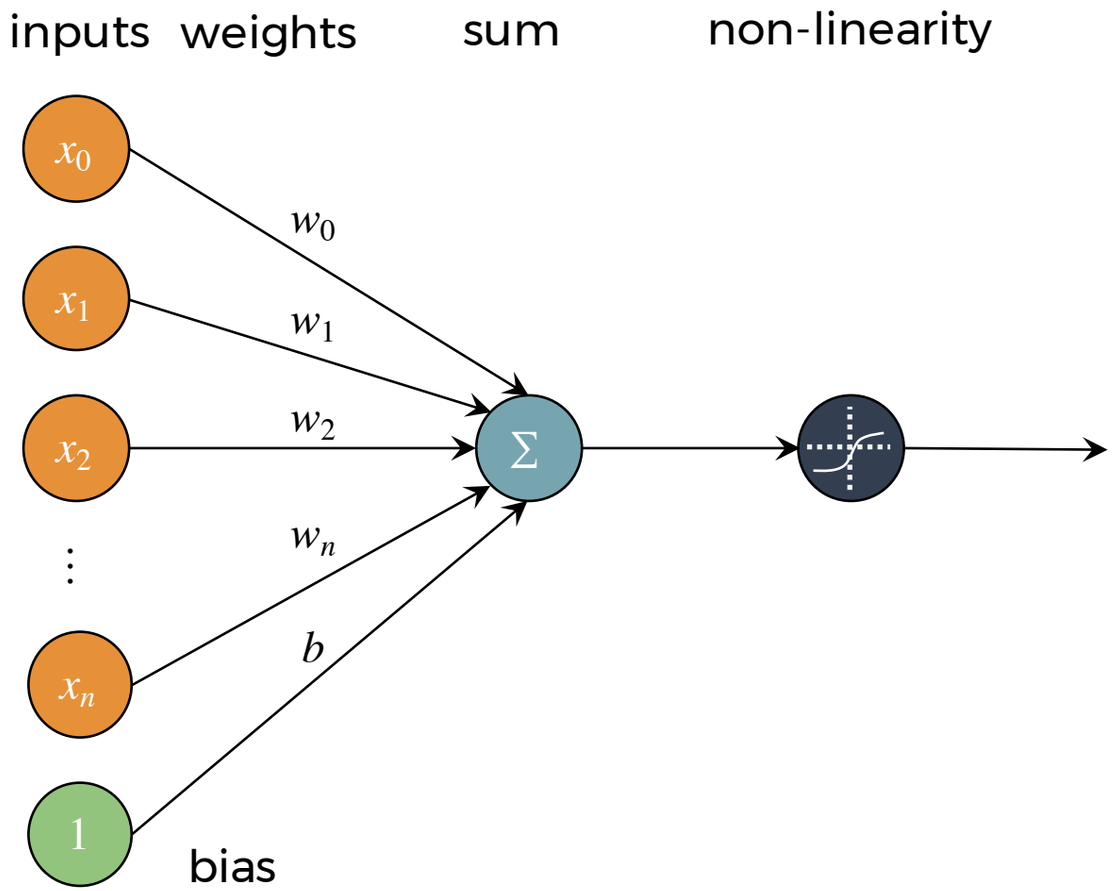
Strictly Increasing

Rectified Linear (ReLU) Activation Function

$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$

$$g(a) = \text{reclin}(a) = \max(0, a)$$





Bounded below by 0 (always non-negative)
Not upper bounded
Strictly increasing
Tends to produce units with sparse activities

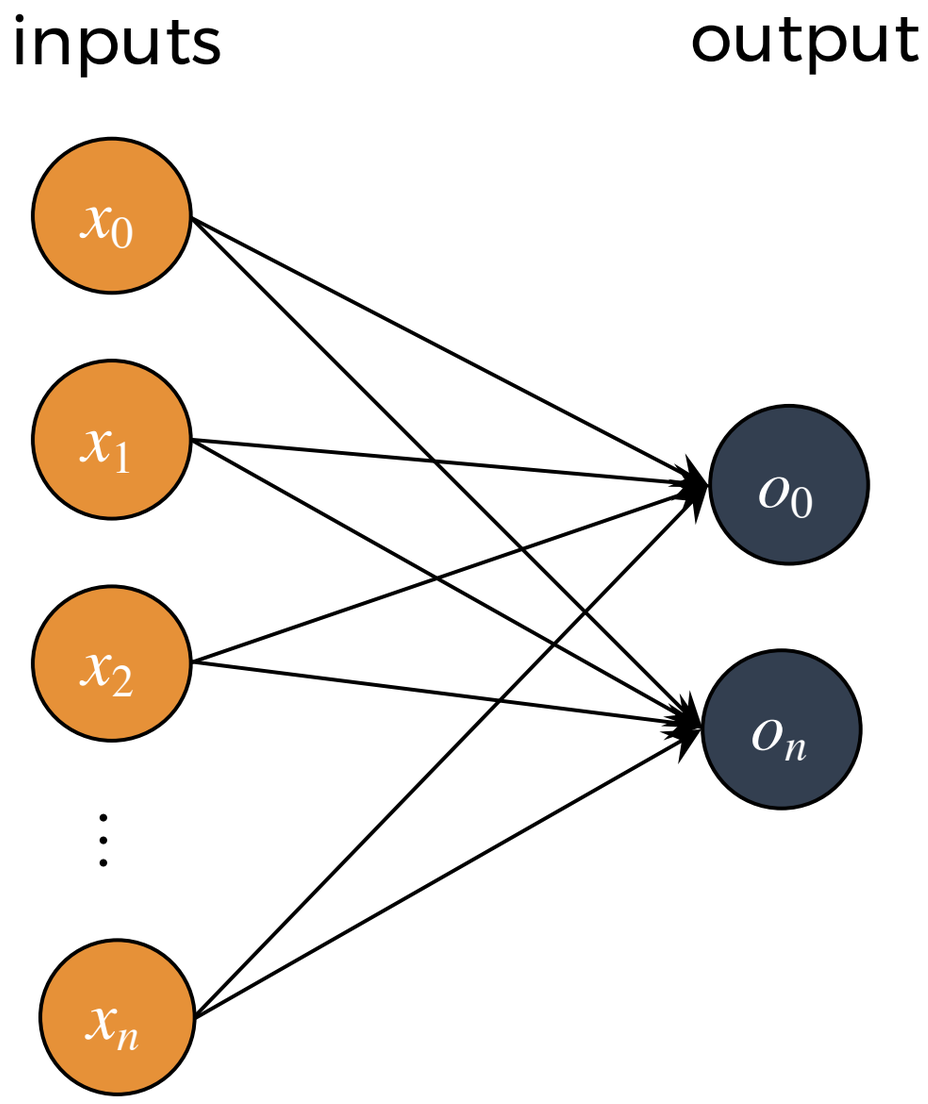
Multi-Output Perceptron

- We need multiple outputs
  (1 output per class)i j
- We need to estimate conditional probability
  p(y = c|x)
- Discriminative Learning

- Softmax activation function at the output

$$\mathbf{o}(\mathbf{a}) = \text{softmax}(\mathbf{a}) = \left[\frac{\exp(a_1)}{\sum_c \exp(a_c)} \cdots \frac{\exp(a_C)}{\sum_c \exp(a_c)}\right]^\top$$

  - Strictly positive
  - sums to one
- Predict class with the highest estimated class conditional probability.



Single Hidden Layer Neural Network

- Hidden layer pre-activation:

$$\mathbf{a}(\mathbf{x}) = \mathbf{b}^{(1)} + \mathbf{W}^{(1)}\mathbf{x}$$

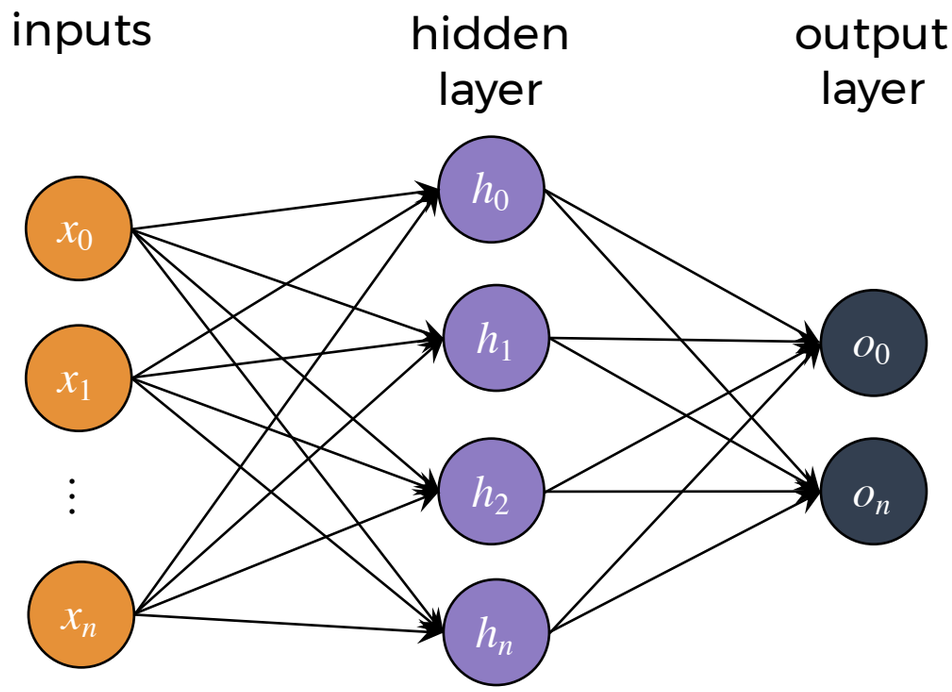
$$\left(a(\mathbf{x})_i = b_i^{(1)} + \sum_j W_{i,j}^{(1)} x_j\right)$$

- Hidden layer activation:

$$\mathbf{h}(\mathbf{x}) = \mathbf{g}(\mathbf{a}(\mathbf{x}))$$

- Output layer activation:

$$\mathbf{o}(\mathbf{x}) = \mathbf{o}\left(b^{(2)} + \mathbf{w}^{(2)}\mathbf{h}^{(1)}\mathbf{x}\right)$$



Multi-Layer Perceptron (MLP)

Consider a network with L hidden layers.

- layer pre-activation for k>0

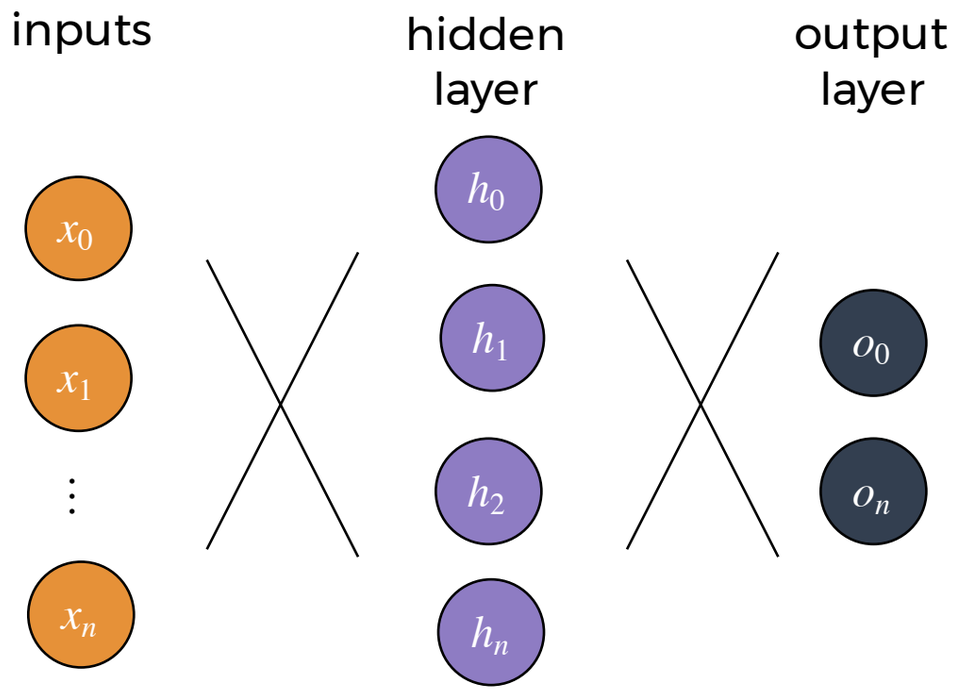
$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x})$$

- hidden layer activation from 1 to L:

$$\mathbf{h}^{(k)}(\mathbf{x}) = \mathbf{g}(\mathbf{a}^{(k)}(\mathbf{x}))$$

- output layer activation (k=L+1)

$$\mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$$



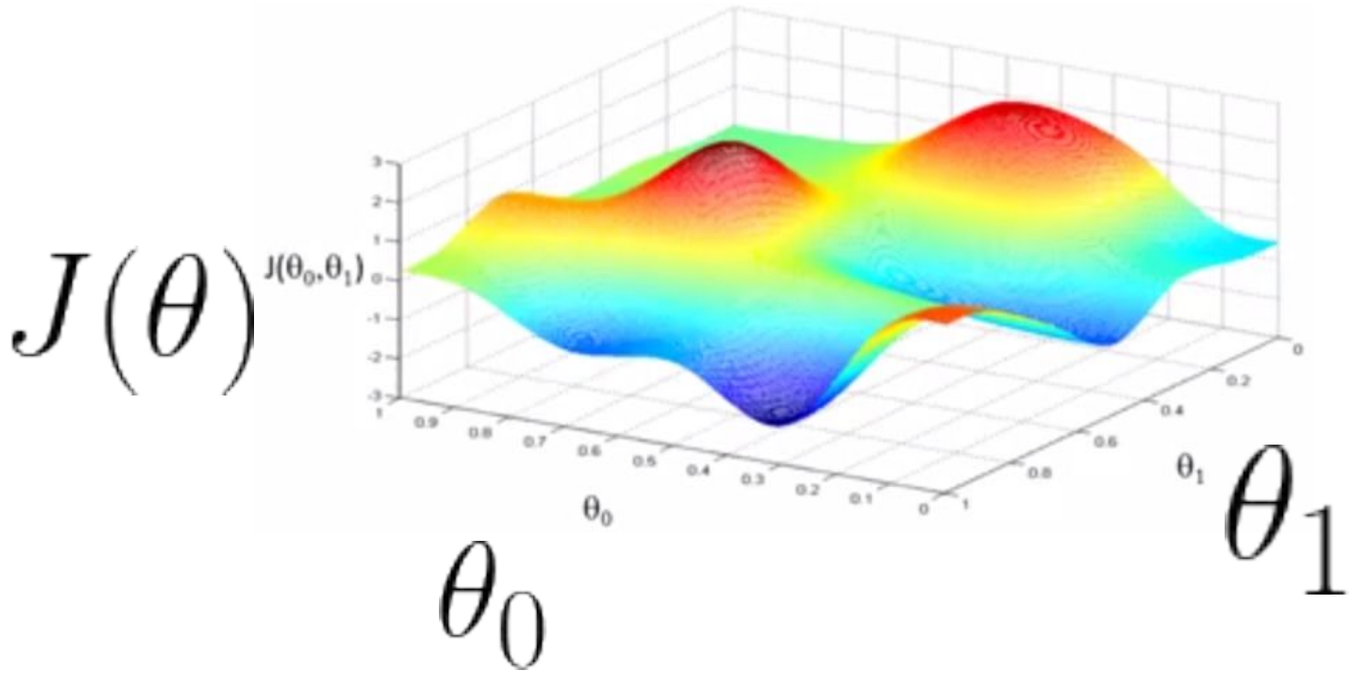
Training

$$J(\theta) = \arg\min_{\boldsymbol{\theta}} \frac{1}{T} \sum_t \underbrace{l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})}_{\text{Loss function}} + \underbrace{\lambda \Omega(\boldsymbol{\theta})}_{\text{Regularizer}}$$

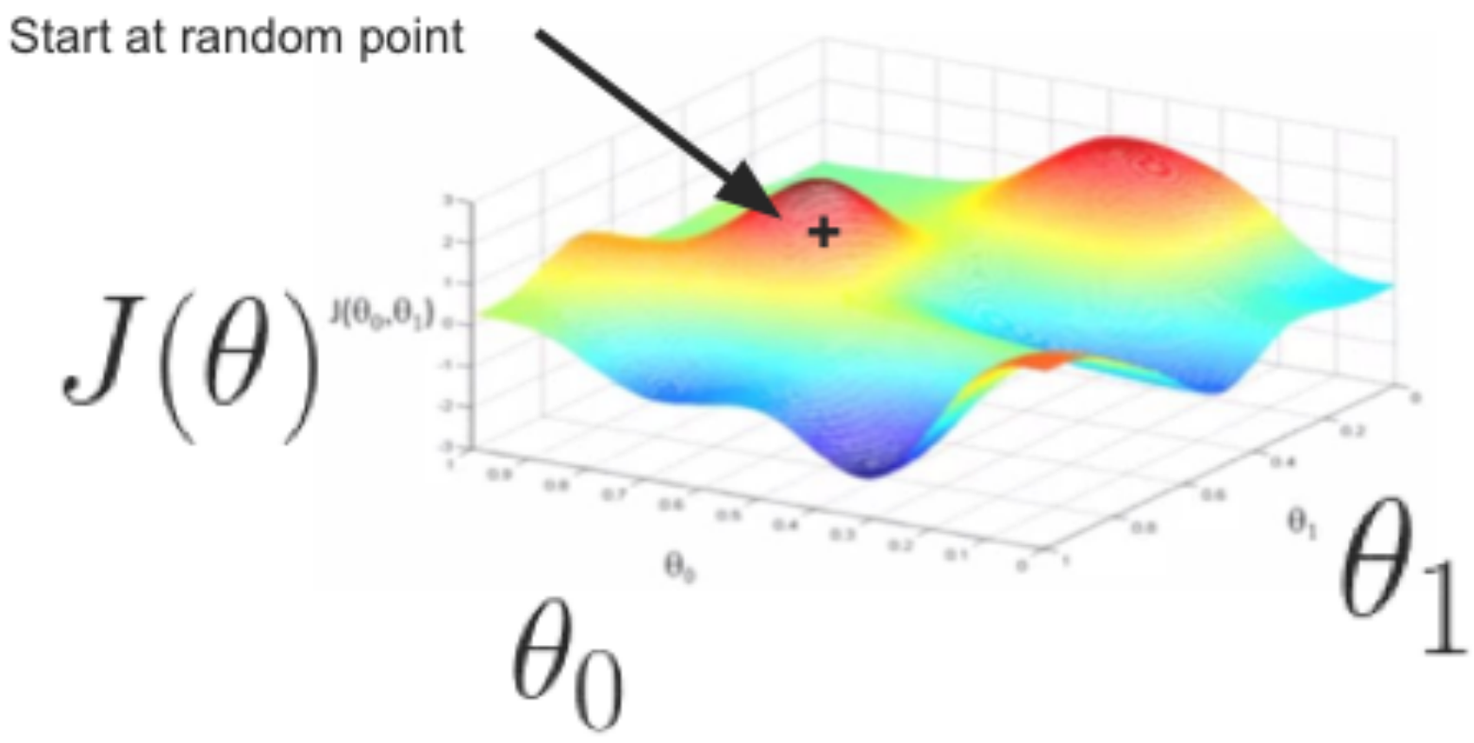
$$\theta = W_1, W_2 ... W_n$$

- Learning is cast as optimization.
- For classification problems, we would like to minimize classification error
- Loss function can sometimes be viewed as a surrogate for what we want to optimize (e.g. upper bound)

Loss is a **function** of the model's parameters

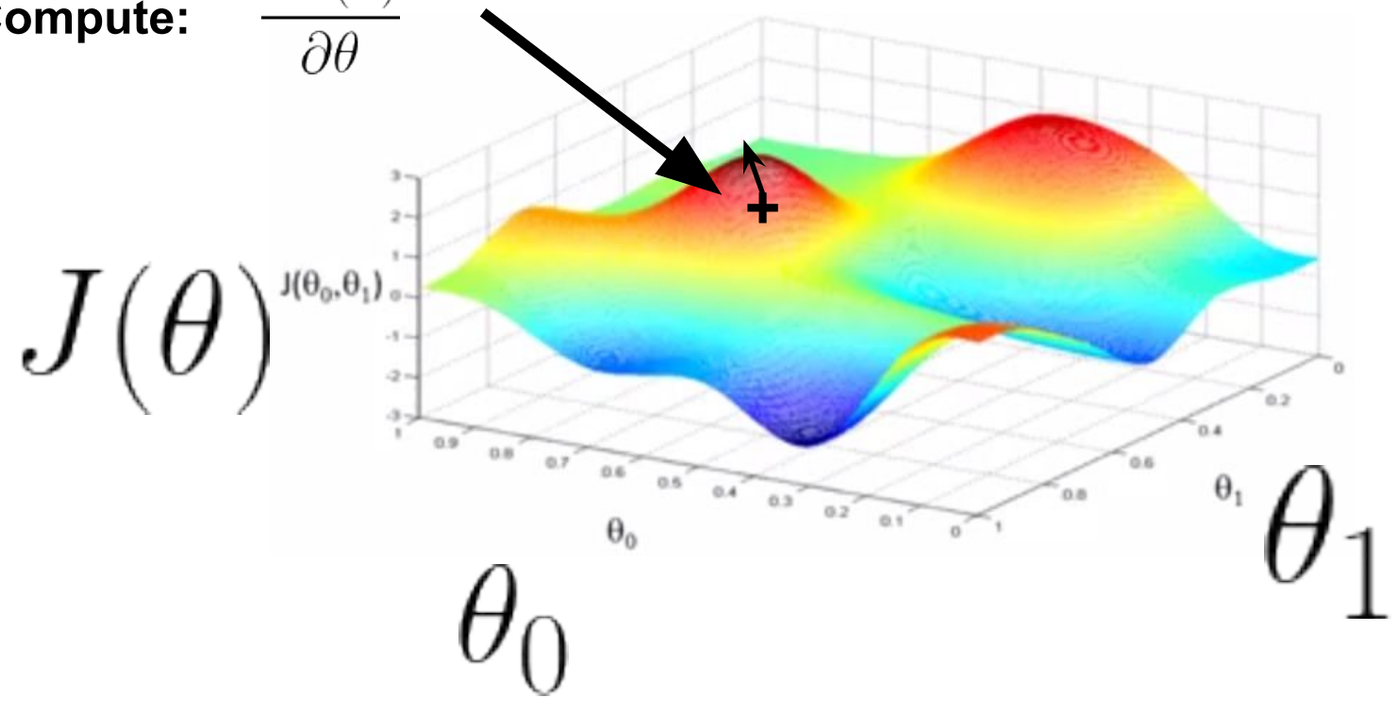


How to minimize loss?



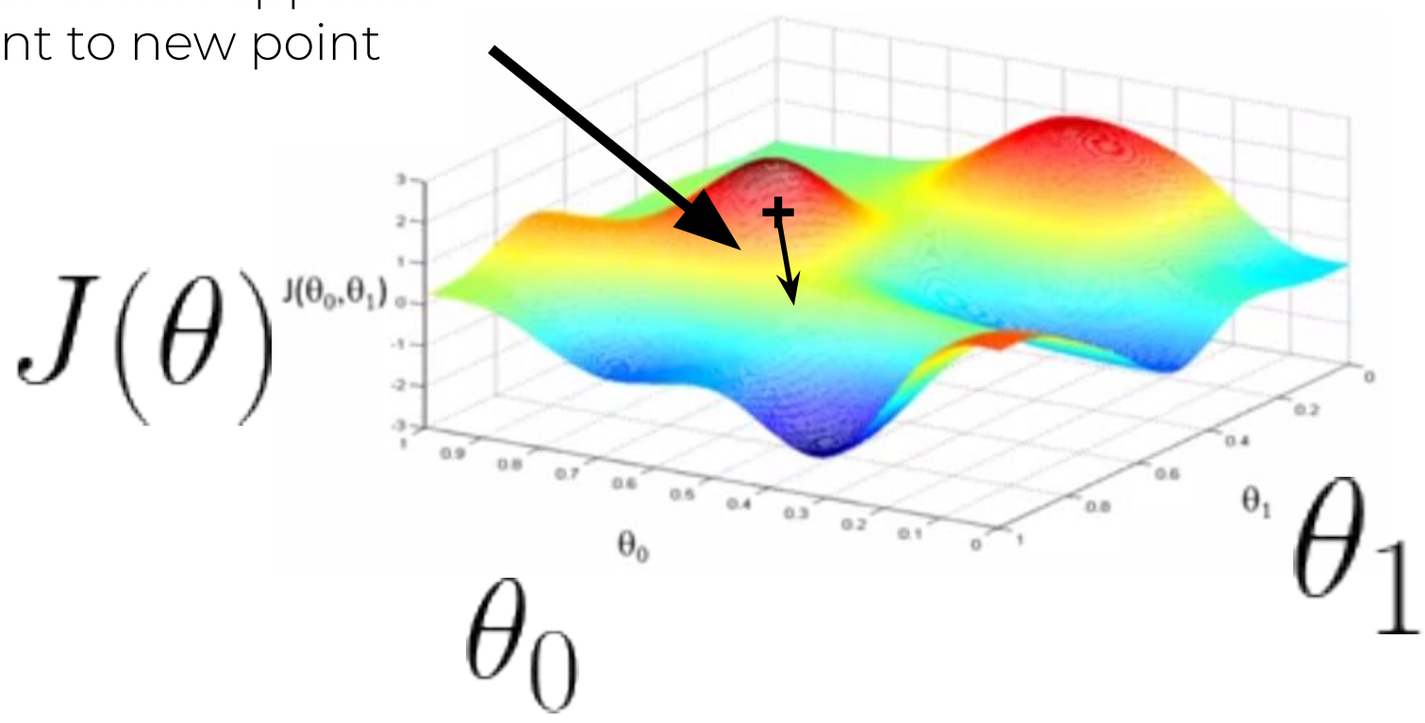
How to minimize loss?

Compute: $\frac{\partial J(\theta)}{\partial \theta}$



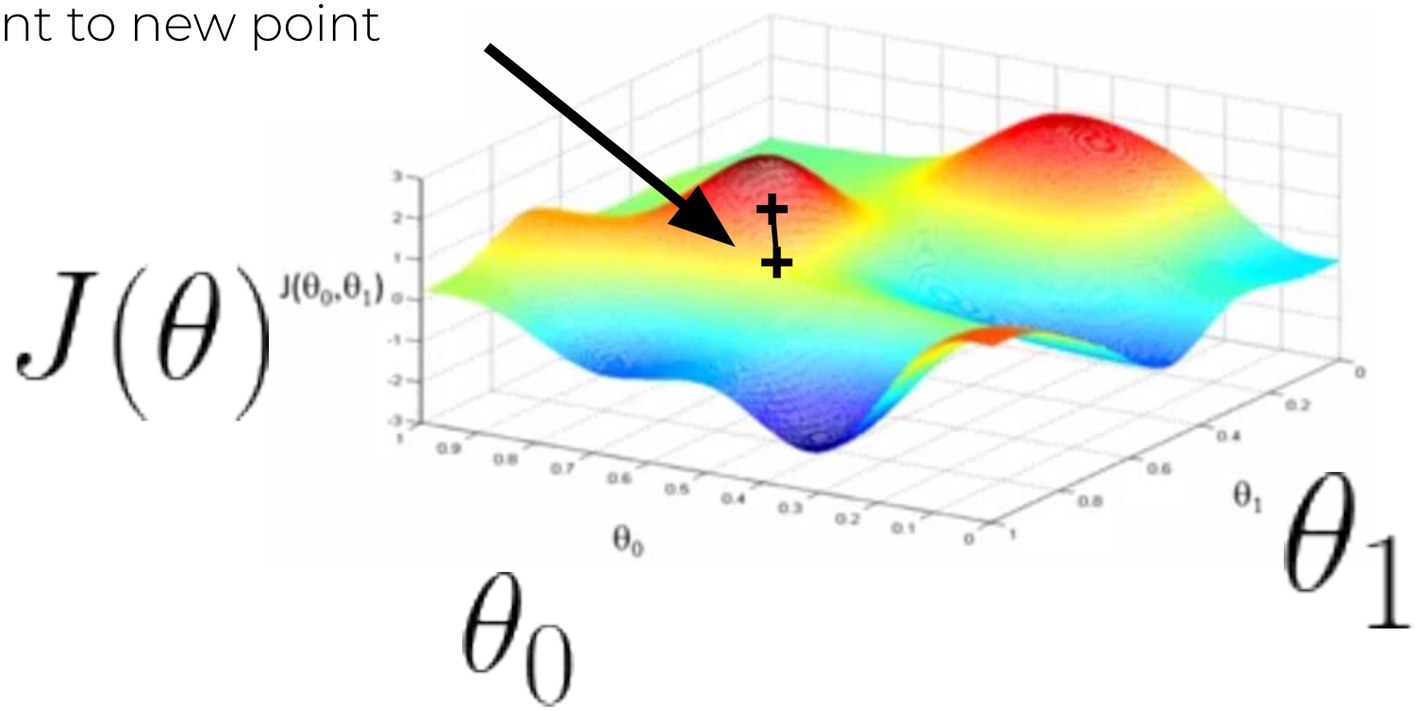
How to minimize loss?

Move in direction opposite of gradient to new point



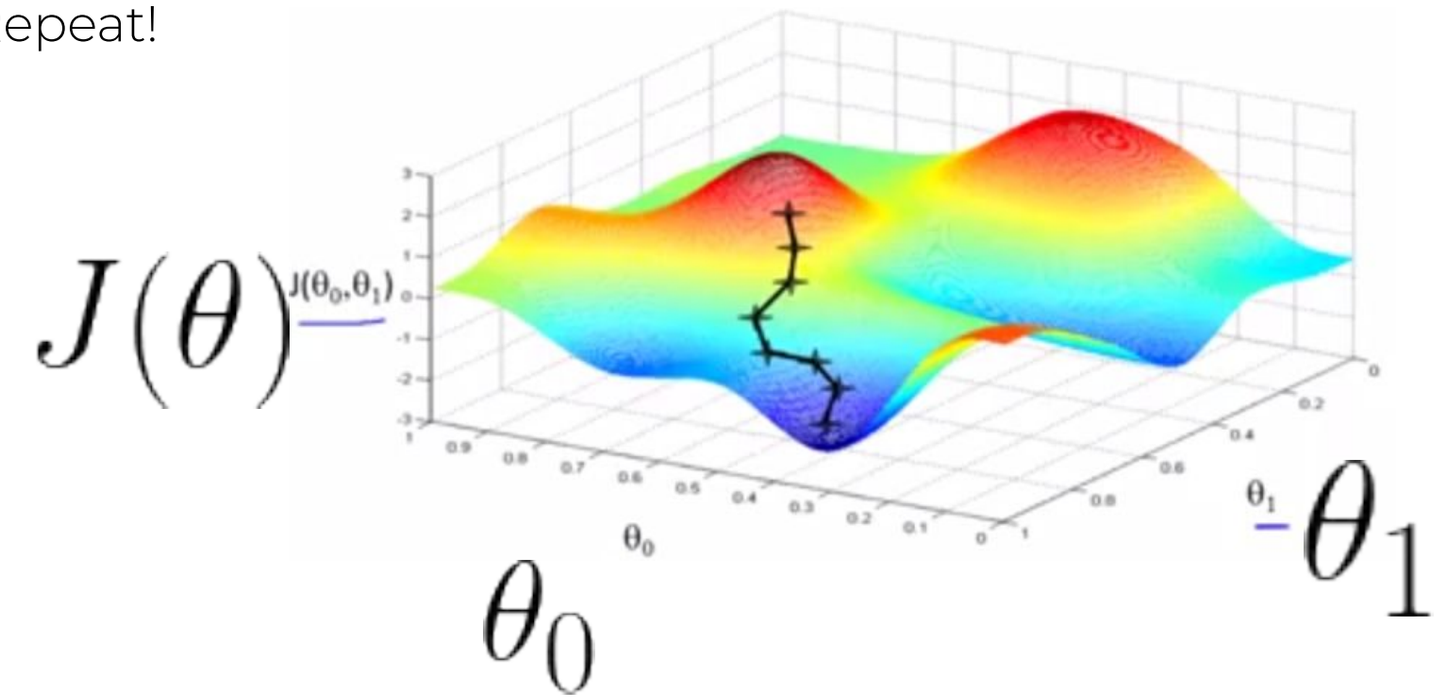
How to minimize loss?

Move in direction opposite of gradient to new point



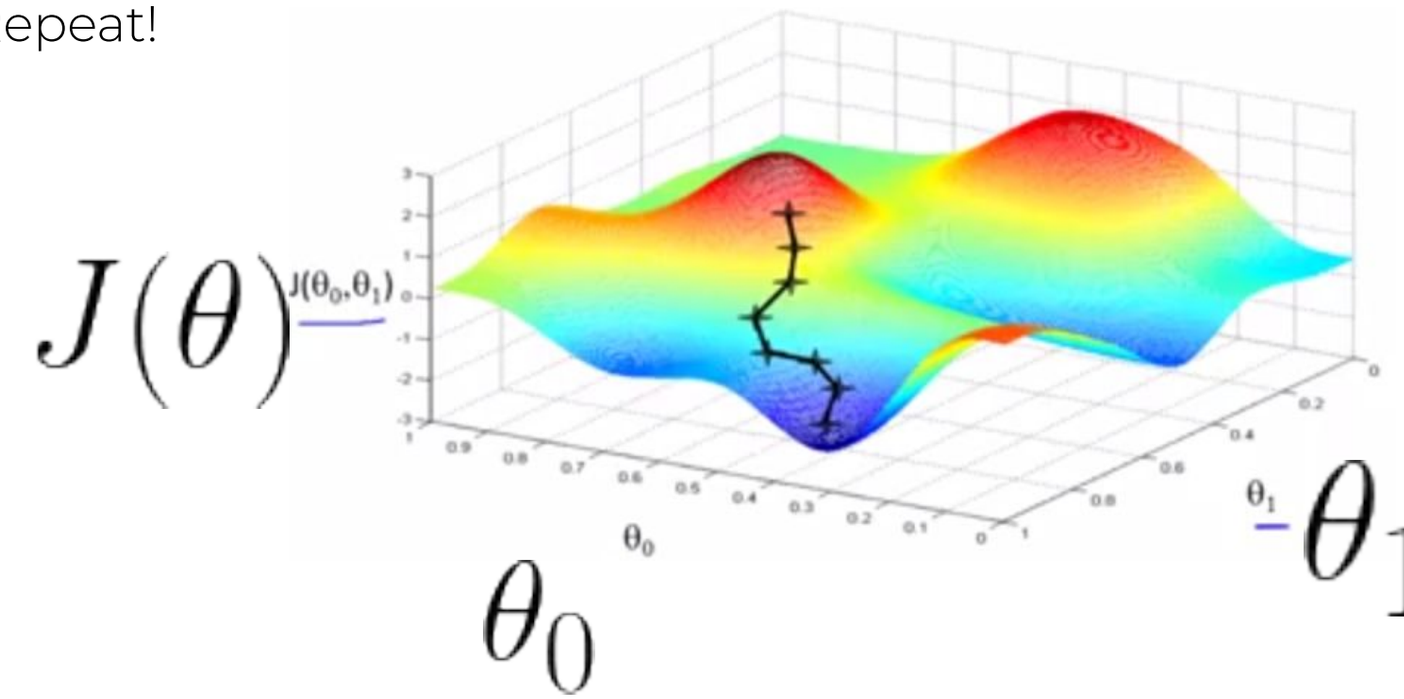
How to minimize loss?

Repeat!



This is called
Stochastic Gradient Descent (SGD)

Repeat!



Stochastic Gradient Descent (SGD)

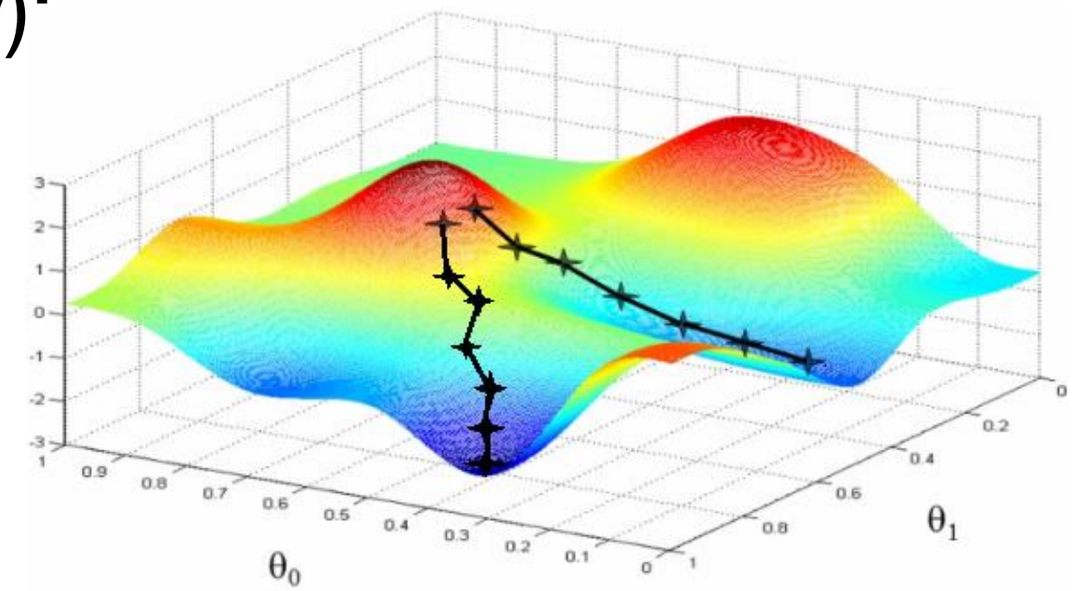
- Initialize θ randomly
- For N Epochs
- For each training example (x, y):

- Compute Loss Gradient:

$$\frac{\partial J(\theta)}{\partial \theta}$$

- Update θ with update rule:

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



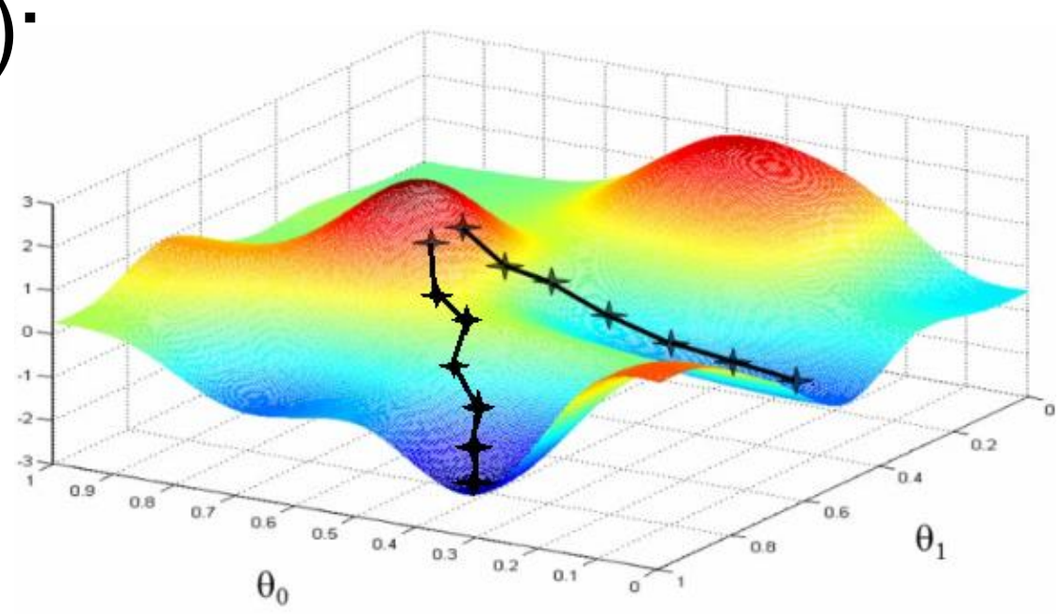
Why is it **Stochastic** Gradient Descent?

- Initialize θ randomly
- For N Epochs
- For each training example (x, y):

- Compute Loss Gradient:

$$\frac{\partial J(\theta)}{\partial \theta}$$

← Only an estimate of true gradient!



- Update θ with update rule:

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

Advantages:

- More accurate estimation of gradient
  - Smoother convergence
  - Allows for larger learning rates
- Minibatches lead to fast training!
  - Can parallelize computation + achieve significant speed increases on GPU's

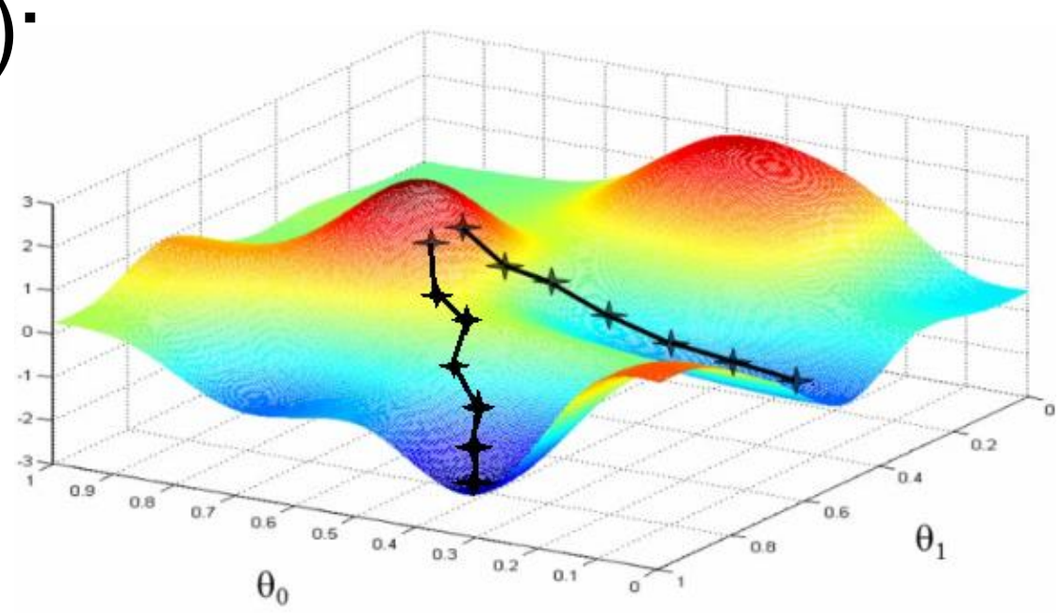
Why is it **Stochastic** Gradient Descent?

- Initialize θ randomly
- For N Epochs
- For each training example (x, y)·

- Compute Loss Gradient:

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{B}\sum_{i}^{B}\frac{\partial J_i(\theta)}{\partial \theta}$$

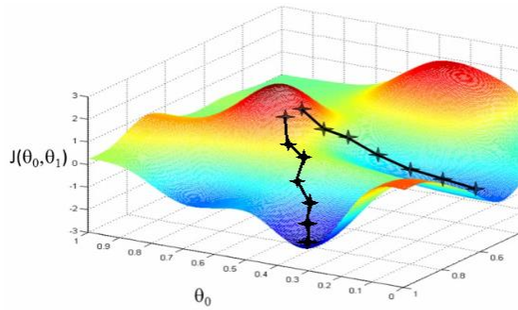
More accurate estimate!



- Update θ with update rule:

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

Stochastic Gradient Descent (SGD)



- Algorithm that performs updates after each example
  - initialize $\boldsymbol{\theta} \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \ldots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$
  - for N iterations
    - for each training example $(\mathbf{x}^{(t)}, y^{(t)})$ or batch

$$\Delta = -\nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) - \lambda \nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \, \Delta$$

Training epoch = Iteration over **all** examples

- To apply this algorithm to neural network training, we need:
  - the loss function $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
  - a procedure to compute the parameter gradients: $\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
  - the regularizer $\Omega(\boldsymbol{\theta})$ (and the gradient $\nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$)

What is a neural network again?

- A family of parametric, non-linear and hierarchical representation learning functions

- $a_L(x; \theta_{1,\ldots,L}) = h_L(h_{L-1}(\ldots h_1(x, \theta_1), \theta_{L-1}), \theta_L)$
- x: input, θl: parameters for layer l, al = hl(x, θl): (non-)linear function

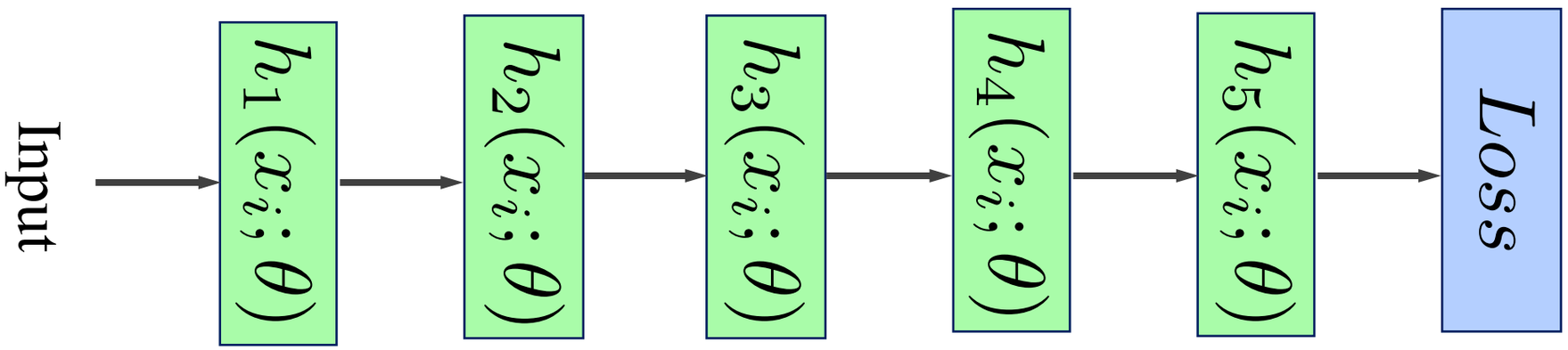
- Given training corpus {X, Y} find optimal parameters

- $\theta^* \leftarrow \arg\min_{\theta} \sum_{(x,y)\in(X,Y)} \ell(y, a_L(x; \theta_{1,\ldots,L}))$

Neural network models

- A neural network model is a series of hierarchically connected functions
- The hierarchy can be very, very complex

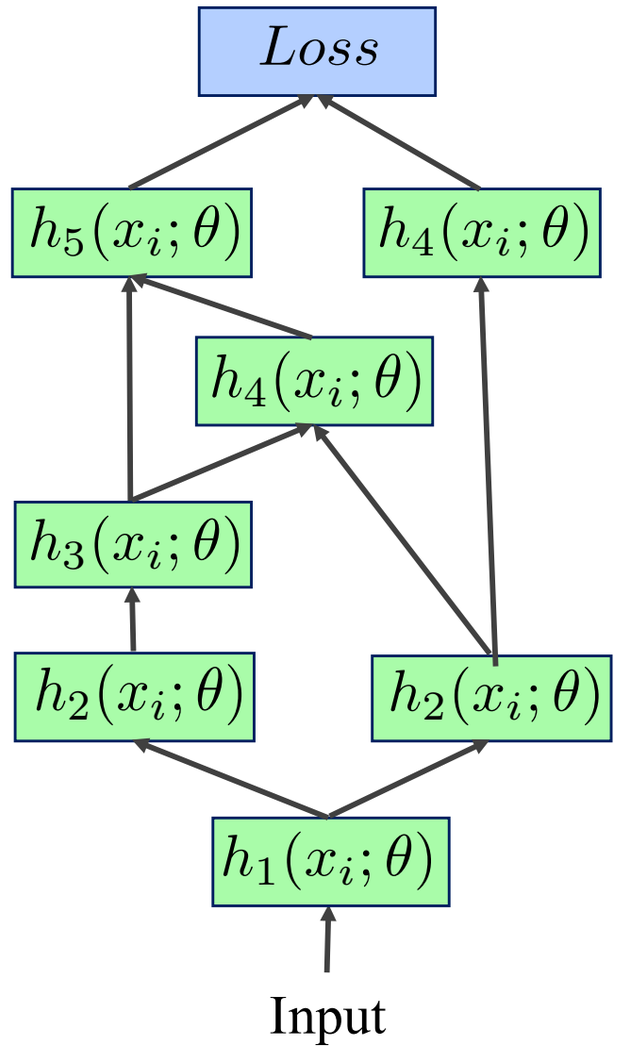
Forward connections (Feedforward architecture)



Neural network models

- A neural network model is a series of hierarchically connected functions
- The hierarchy can be very, very complex

Interweaved connections (Directed Acyclic Graph architecture – DAGNN)



Again, what is a neural network again?

- $a_L(x; \theta_{1,\ldots,L}) = h_L(h_{L-1}(\ldots h_1(x, \theta_1), \theta_{L-1}), \theta_L)$
- x: input, θl: parameters for layer l, al = hl(x, θl): (non-)linear function
- Given training corpus {X, Y} find optimal parameters

$$\theta^* \leftarrow \arg\min_{\theta} \sum_{(x,y)\in(X,Y)} \ell(y, a_L(x; \theta_{1,\ldots,L}))$$

- To use gradient descent optimization
- we need the gradients $\left(\theta^{t+1} = \theta^t - \eta_t \frac{\partial \mathcal{L}}{\partial \theta^t}\right)$

$$\frac{\partial \mathcal{L}}{\partial \theta_l}, l = 1, \ldots, L$$

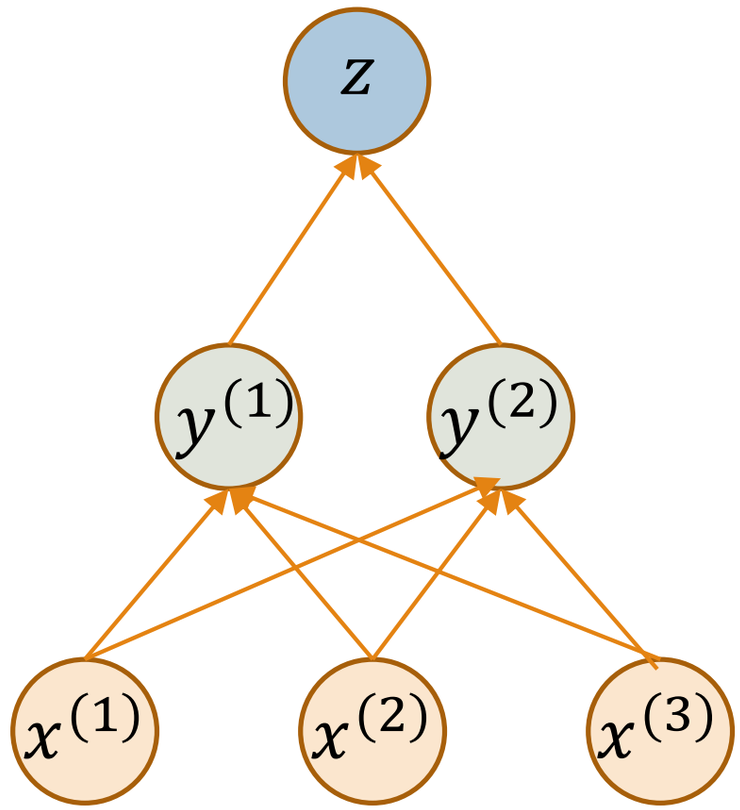
- How to compute the gradients for such a complicated function enclosing other functions, like aL (...)?

Chain rule

- Assume a nested function, $z = f(y)$ and $y = g(x)$

- Chain Rule for scalars x, y, z

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$



- When

$$\frac{dz}{dx^i} = \sum_j \frac{dz}{dy^j}\frac{dy^j}{dx^i}$$

gradients from all possible paths

Chain rule

- Assume a nested function, $z = f(y)$ and $y = g(x)$

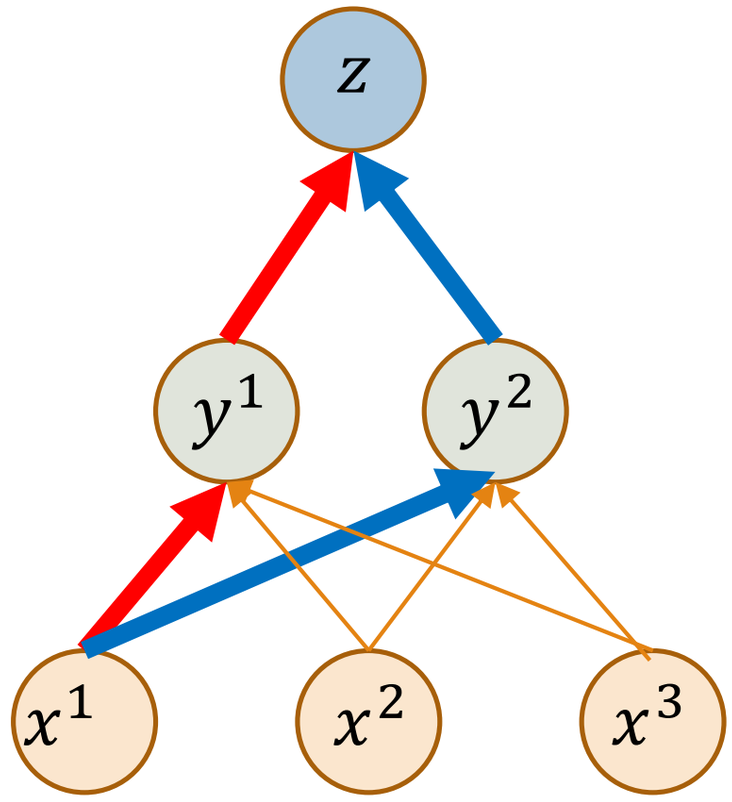
- Chain Rule for scalars x, y, z

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

- When

$$\frac{dz}{dx^i} = \sum_j \frac{dz}{dy^j}\frac{dy^j}{dx^i}$$

gradients from all possible paths



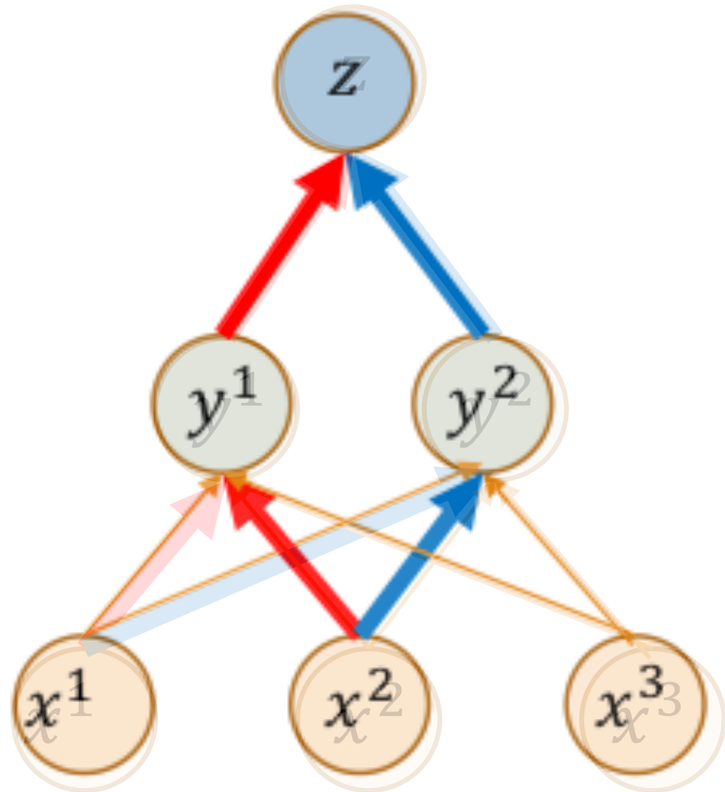
$$\frac{dz}{dx^1} = \frac{dz}{dy^1}\frac{dy^1}{dx^1} + \frac{dz}{dy^2}\frac{dy^2}{dx^1}$$

Chain rule

- Assume a nested function, $z = f(y)$ and $y = g(x)$

- Chain Rule for scalars x, y, z

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$



- When

$$\frac{dz}{dx^i} = \sum_j \frac{dz}{dy^j}\frac{dy^j}{dx^i}$$

gradients from all possible paths

$$\frac{dz}{dx^2} = \frac{dz}{dy^1}\frac{dy^1}{dx^2} + \frac{dz}{dy^2}\frac{dy^2}{dx^2}$$

Chain rule

- Assume a nested function, $z = f(y)$ and $y = g(x)$

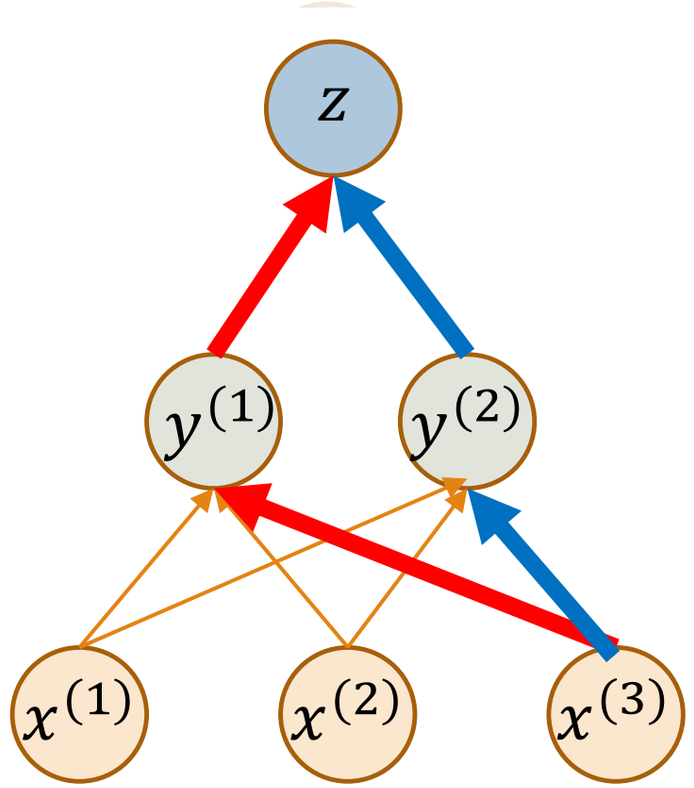
- Chain Rule for scalars x, y, z

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

- When

$$\frac{dz}{dx^i} = \sum_j \frac{dz}{dy^j}\frac{dy^j}{dx^i}$$

gradients from all possible paths



$$\frac{dz}{dx^3} = \frac{dz}{dy^1}\frac{dy^1}{dx^3} + \frac{dz}{dy^2}\frac{dy^2}{dx^3}$$

Backpropagation $\Longleftrightarrow$ Chain rule!!!

- The loss function $\mathcal{L}(y, a_L)$ depends on aL, which depends on aL-1 , …, which depends on a2:

$$a_L(x; \theta_{1,\ldots,L}) = h_L(h_{L-1}(\ldots h_1(x, \theta_1), \theta_{L-1}), \theta_L)$$

- Gradients of parameters of layer l → Chain rule

$$\frac{\partial \mathcal{L}}{\partial \theta_l} = \frac{\partial \mathcal{L}}{\partial a_l} \cdot \frac{\partial a_L}{\partial a_{L-1}} \cdot \frac{\partial a_{L-1}}{\partial a_{L-2}} \cdot \ldots \cdot$$

- When shortened, we need to two quantities

$$\frac{\partial \mathcal{L}}{\partial \theta_l} = \left(\frac{\partial a_l}{\partial \theta_l}\right)^T \cdot \frac{\partial \mathcal{L}}{\partial a_l}$$

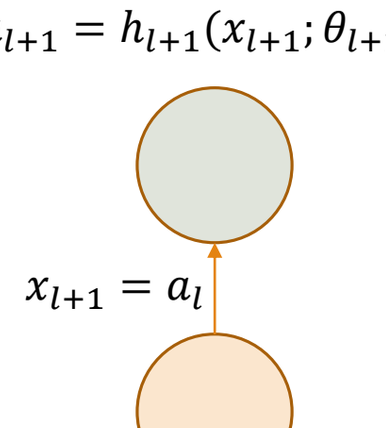
Gradient of a module w.r.t. its parameters

Gradient of loss w.r.t. the module output

Backpropagation $\Longleftrightarrow$ Chain rule!!!

- For $\frac{\partial \mathcal{L}}{\partial a_l}$ in $\frac{\partial \mathcal{L}}{\partial \theta_l} = \left(\frac{\partial a_l}{\partial \theta_l}\right)^T \cdot \frac{\partial \mathcal{L}}{\partial a_l}$ we apply chain rule again

$$\frac{\partial \mathcal{L}}{\partial a_l} = \left(\frac{\partial a_{l+1}}{\partial a_l}\right)^T \cdot \frac{\partial \mathcal{L}}{\partial a_{l+1}}$$



- We can rewrite $\frac{\partial a_{l+1}}{\partial a_l}$ as gradient of module w.r.t. to input
- Remember, the output of a module is the input for the next one:

$$\frac{\partial \mathcal{L}}{\partial a_l} = \left(\frac{\partial a_{l+1}}{\partial x_{l+1}}\right)^T \cdot \frac{\partial \mathcal{L}}{\partial a_{l+1}}$$

Recursive rule

So what is deep learning?

Three key ideas

- (Hierarchical) Compositionality
- End-to-End Learning
- Distributed Representations

slide by Dhruv Batra

Three key ideas

- **(Hierarchical) Compositionality**
  - Cascade of non-linear transformations
  - Multiple layers of representations
- End-to-End Learning
  - Learning (goal-driven) representations
  - Learning to feature extract
- Distributed Representations
  - No single neuron “encodes” everything
  - Groups of neurons work together

slide by Dhruv Batra

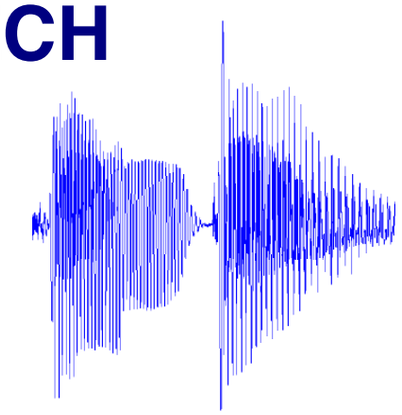
Traditional Machine Learning

VISION

hand-crafted features SIFT/HOG → your favorite classifier → "car"

fixed | learned

SPEECH



hand-crafted features MFCC → your favorite classifier → \ˈd ē p\

fixed | learned

NLP

This burrito place is yummy and fun!

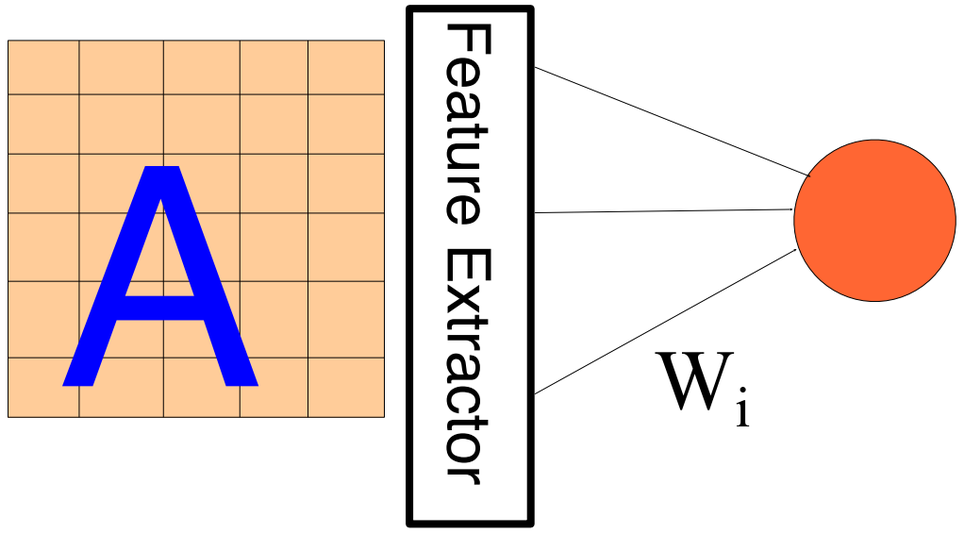
hand-crafted features Bag-of-words → your favorite classifier → "+"

fixed | learned

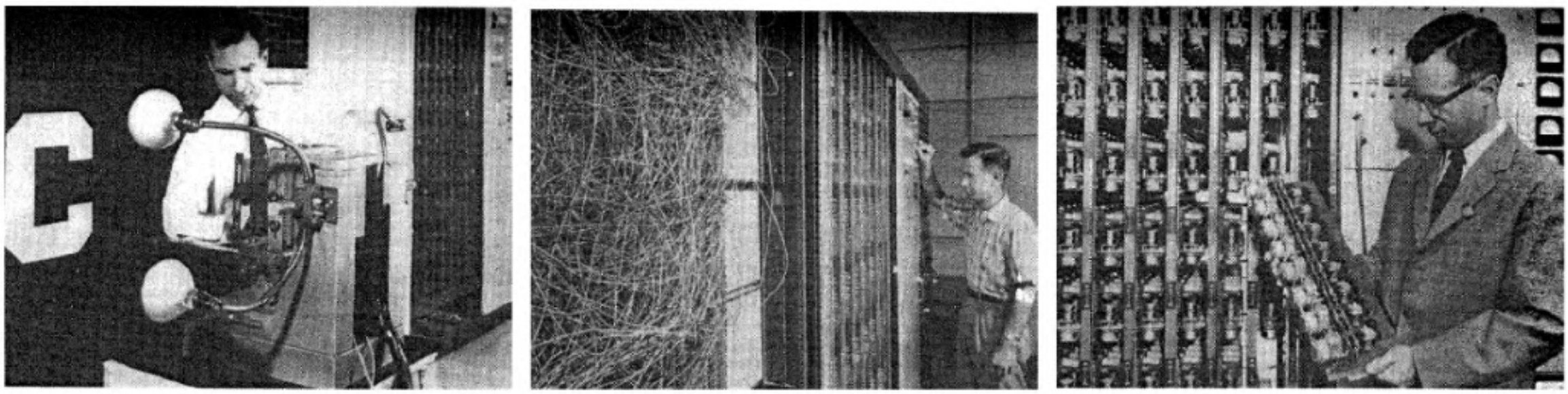
slide by Marc'Aurelio Ranzato, Yann LeCun

It’s an old paradigm

- The first learning machine: the Perceptron
  - Built at Cornell in 1960
- The Perceptron was a linear classifier on top of a simple feature extractor
- The vast majority of practical applications of ML today use glorified linear classifiers or glorified template matching.
- Designing a feature extractor requires considerable efforts by experts.



$$y = sign\left(\sum_{i=1}^{N} W_i F_i(X) + b\right)$$



slide by Marc’Aurelio Ranzato, Yann LeCun

Hierarchical Compositionality

VISION

pixels → edge → texton → motif → part → object

SPEECH

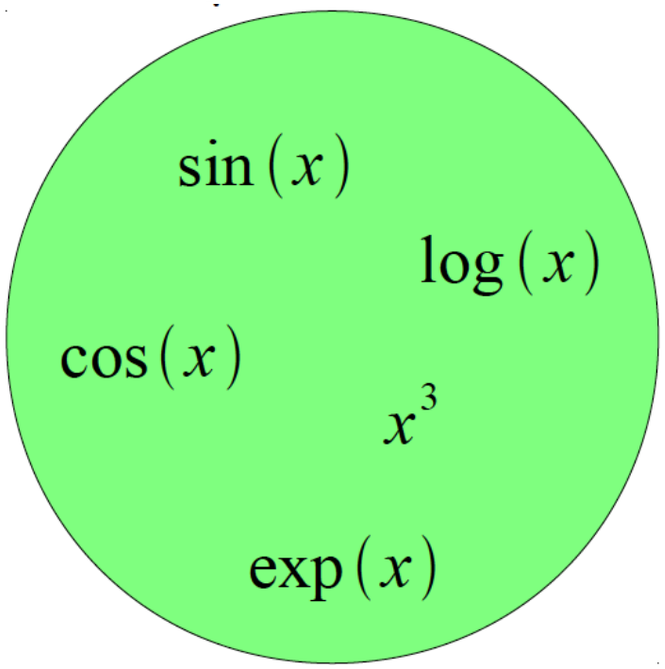
sample → spectral band → formant → motif → phone → word

NLP

character → word → NP/VP/.. → clause → sentence → story

Building A Complicated Function

Given a library of simple functions

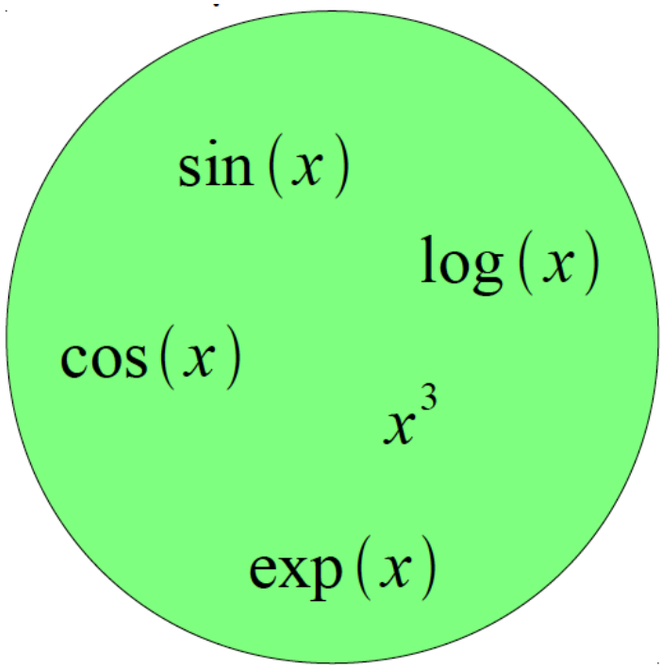


Compose into a

complicate function

Building A Complicated Function

Given a library of simple functions



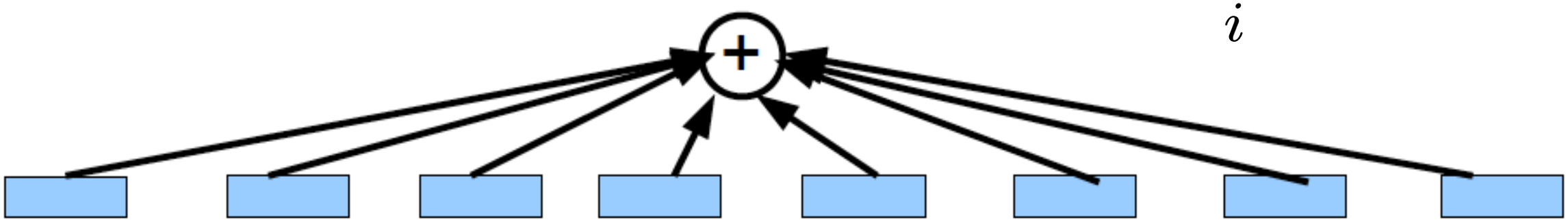
Compose into a

complicate function

Idea 1: Linear Combinations

- Boosting
- Kernels
- …

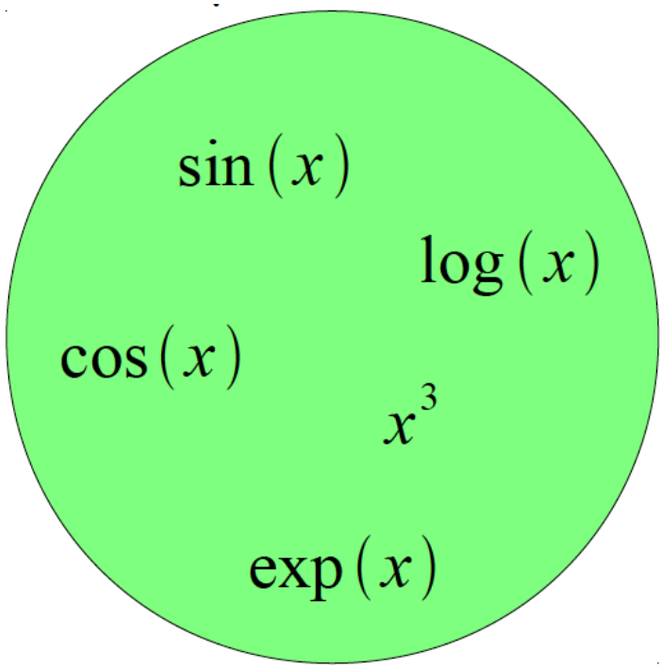
$$f(x) = \sum_i \alpha_i g_i(x)$$



slide by Marc'Aurelio Ranzato, Yann LeCun

Building A Complicated Function

Given a library of simple functions



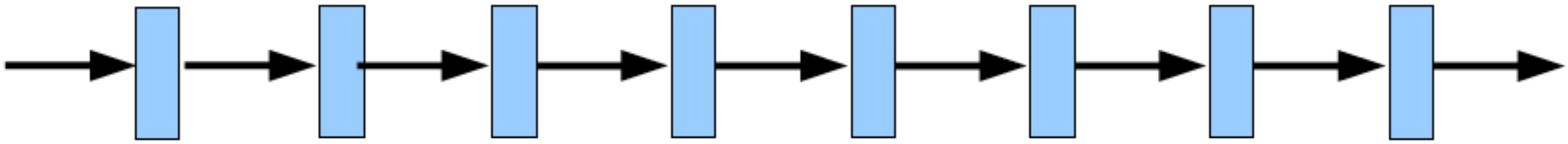
Compose into a

complicate function

Idea 2: Compositions

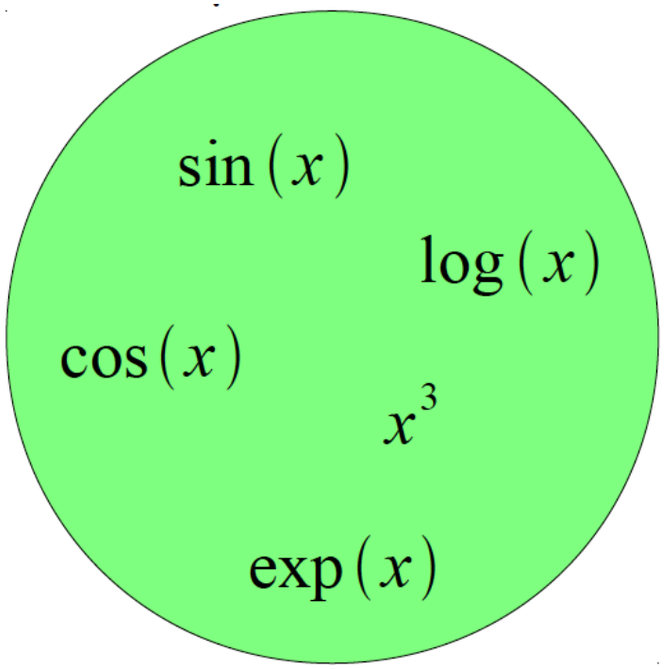
- Deep Learning
- Grammar models
- Scattering transforms…

$$f(x) = g_1(g_2(\dots(g_n(x)\dots))$$



Building A Complicated Function

Given a library of simple functions



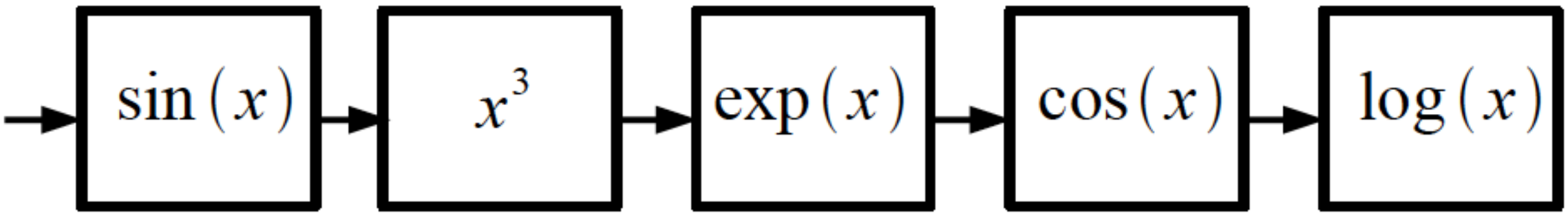
Compose into a

complicate function

Idea 2: Compositions

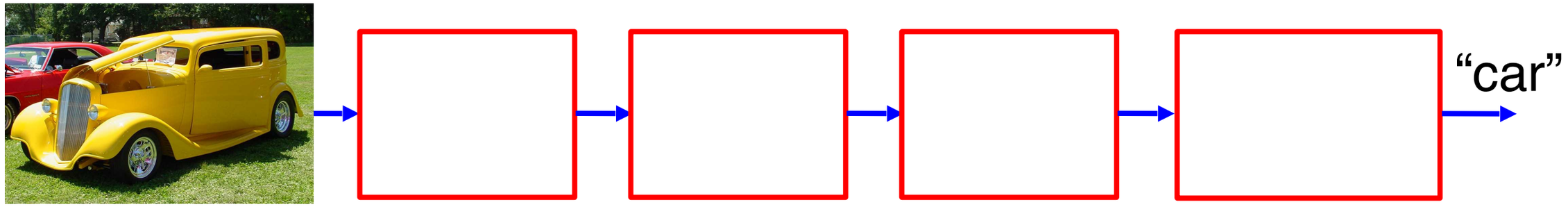
- Deep Learning
- Grammar models
- Scattering transforms…

$$f(x) = \log(\cos(\exp(\sin^3(x))))$$



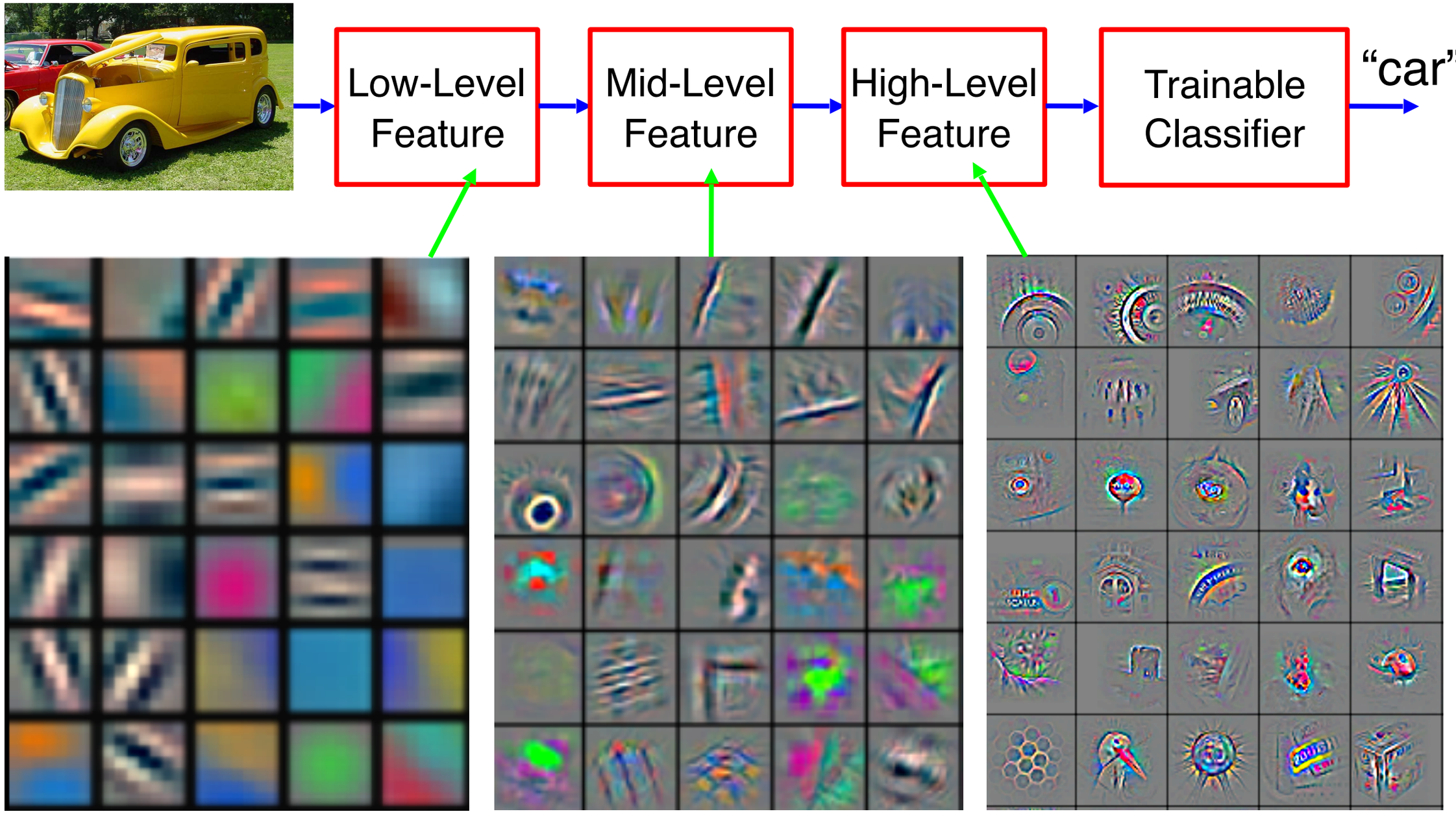
slide by Marc'Aurelio Ranzato, Yann LeCun

Deep Learning = Hierarchical Compositionality



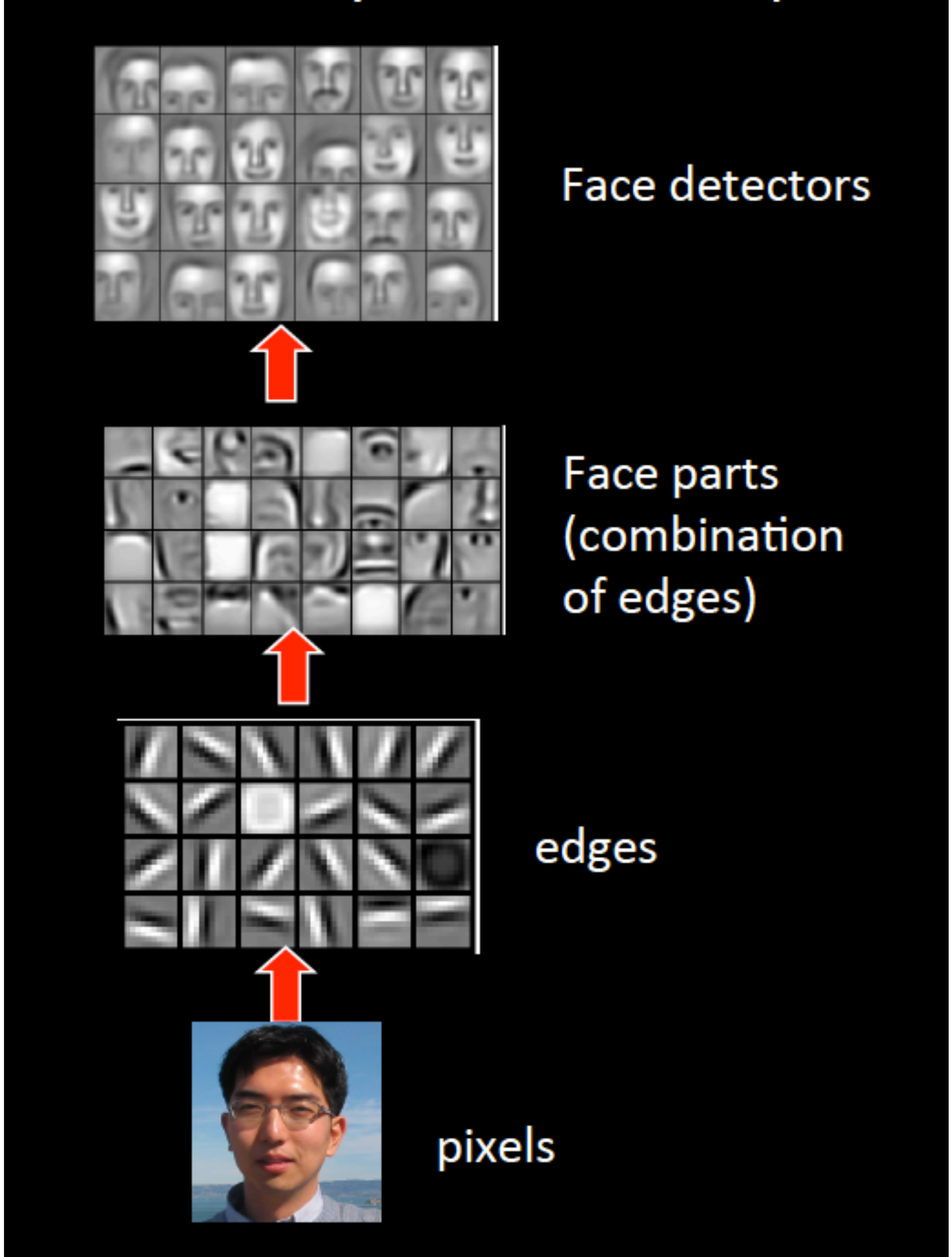
slide by Marc'Aurelio Ranzato, Yann LeCun

Deep Learning = Hierarchical Compositionality



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

slide by Marc’Aurelio Ranzato, Yann LeCun



Sparse DBNs
[Lee et al. ICML '09]
Figure courtesy: Quoc Le

slide by Dhruv Batra

Three key ideas

- (Hierarchical) Compositionality
  - Cascade of non-linear transformations
  - Multiple layers of representations

- **End-to-End Learning**
  - Learning (goal-driven) representations
  - Learning to feature extract

- Distributed Representations
  - No single neuron “encodes” everything
  - Groups of neurons work together

slide by Dhruv Batra

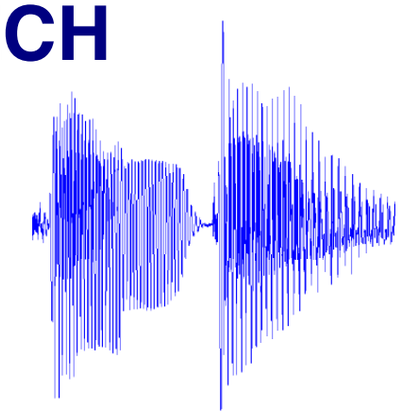
Traditional Machine Learning

VISION

hand-crafted features SIFT/HOG → your favorite classifier → "car"

fixed | learned

SPEECH



hand-crafted features MFCC → your favorite classifier → \ˈd ē p\

fixed | learned

NLP

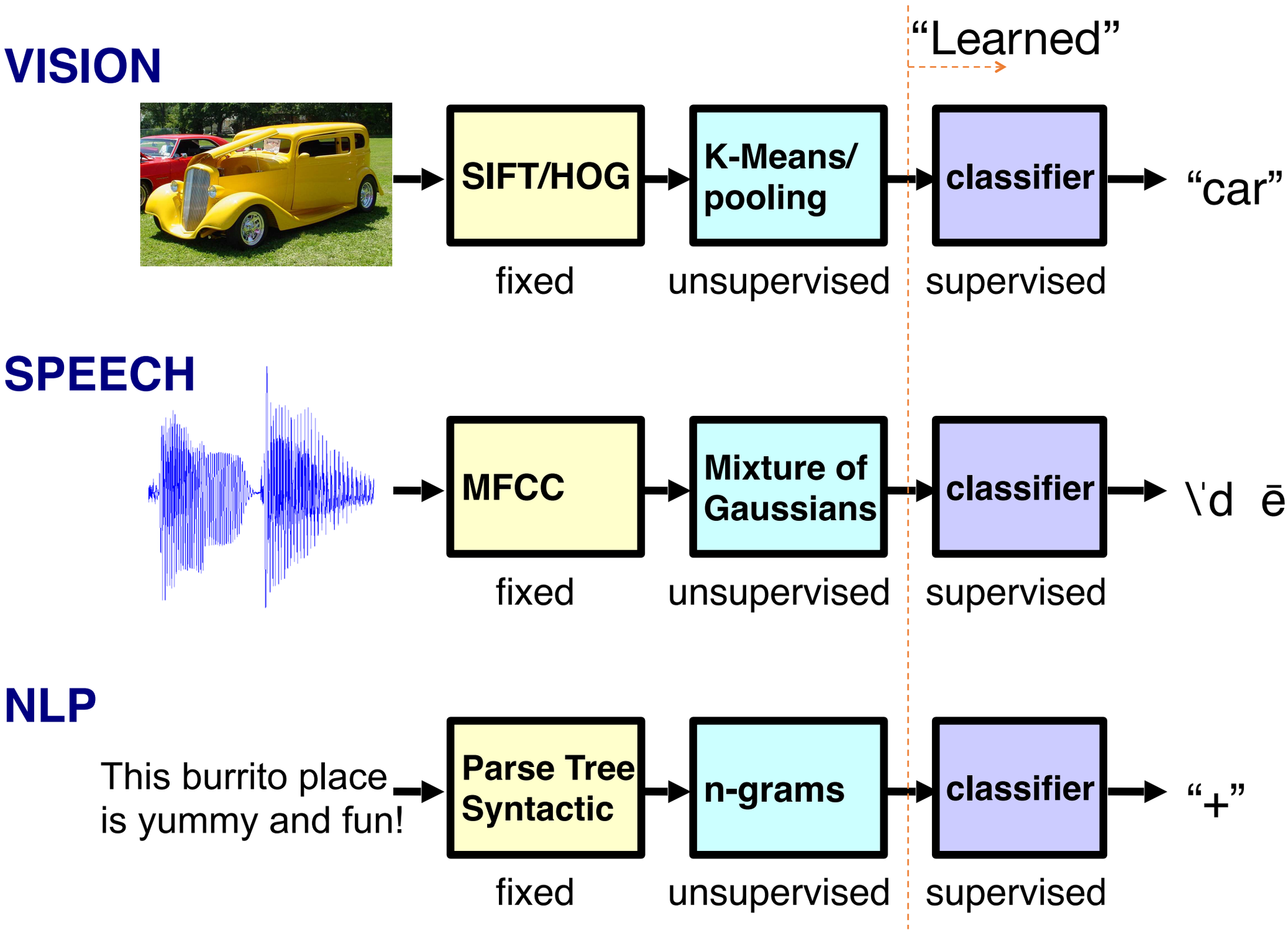
This burrito place is yummy and fun!

hand-crafted features Bag-of-words → your favorite classifier → "+"

fixed | learned

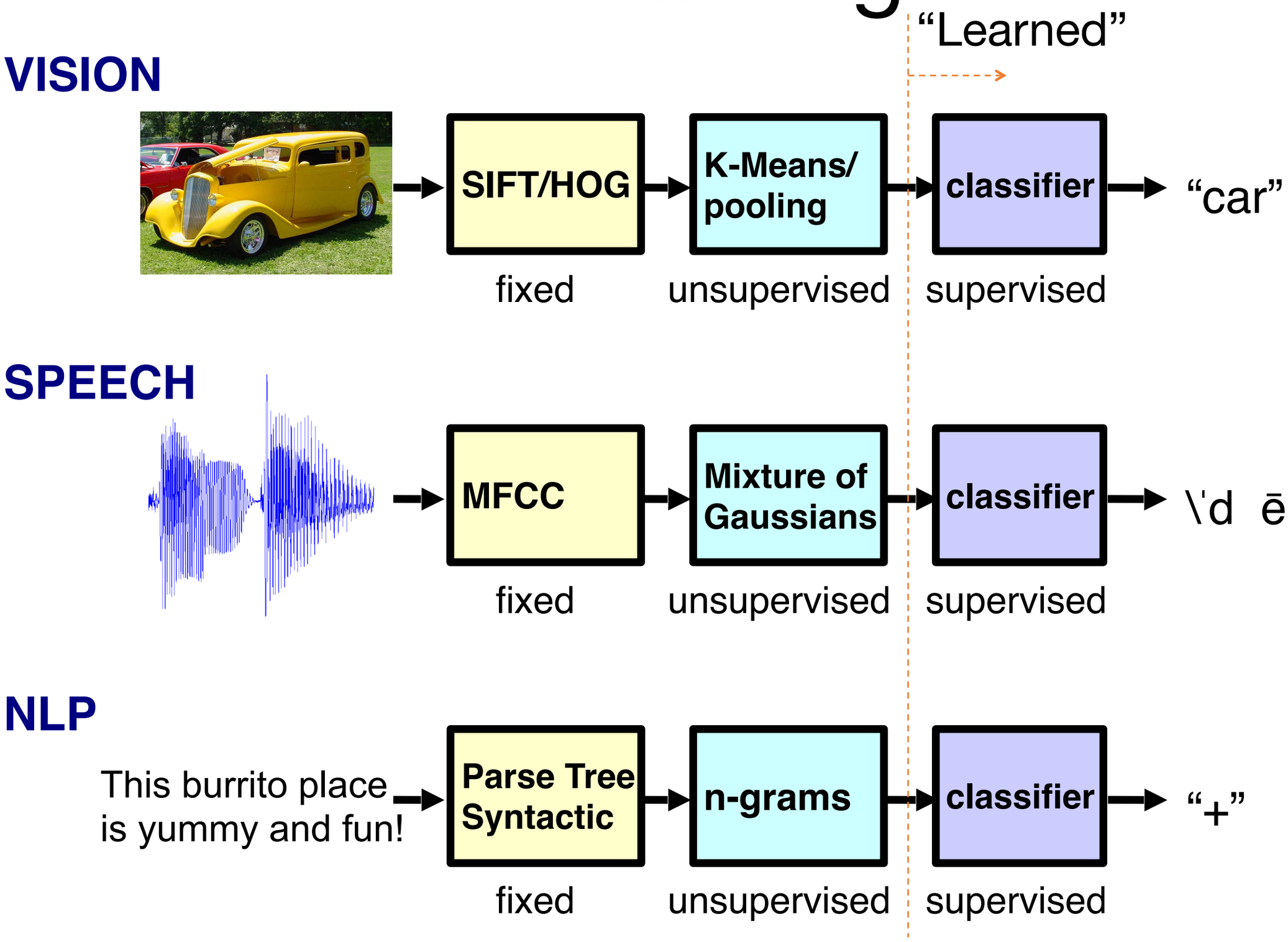
slide by Marc'Aurelio Ranzato, Yann LeCun

Traditional Machine Learning (more accurately)



slide by Marc'Aurelio Ranzato, Yann LeCun

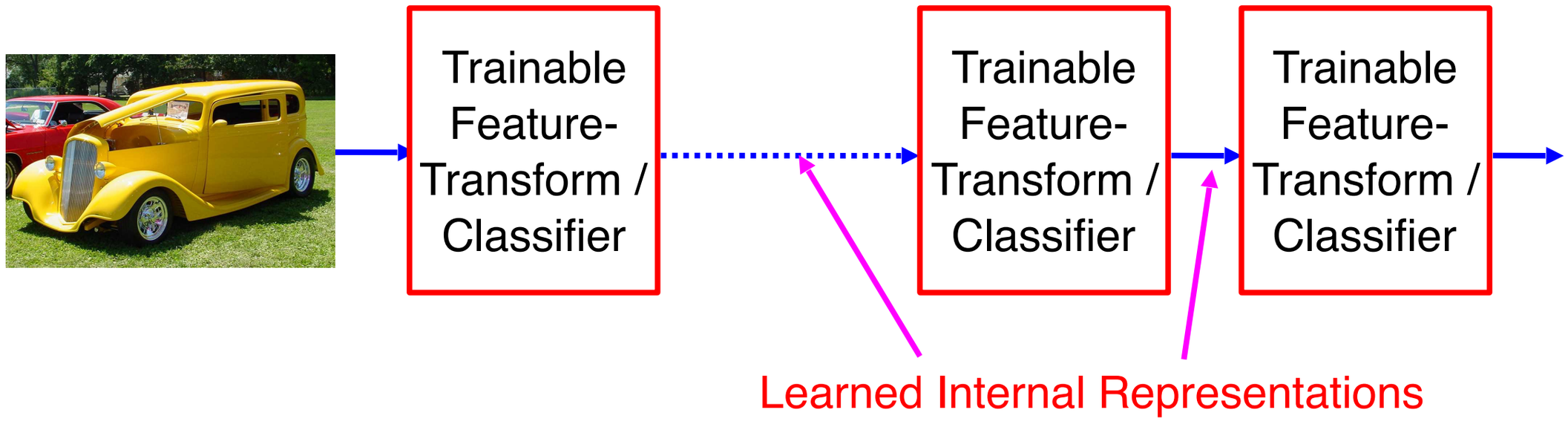
Deep Learning = End-to-End Learning



slide by Marc'Aurelio Ranzato, Yann LeCun

Deep Learning = End-to-End Learning

- A hierarchy of trainable feature transforms
  - Each module transforms its input representation into a higher-level one.
  - High-level features are more global and more invariant
  - Low-level features are shared among categories



slide by Marc'Aurelio Ranzato, Yann LeCun

“Shallow” vs Deep Learning

- “Shallow” models

hand-crafted
Feature Extractor

fixed

“Simple” Trainable
Classifier

learned

- Deep models

Trainable
Feature-
Transform /
Classifier

Trainable
Feature-
Transform /
Classifier

Trainable
Feature-
Transform /
Classifier

Learned Internal Representations

slide by Marc’Aurelio Ranzato, Yann LeCun

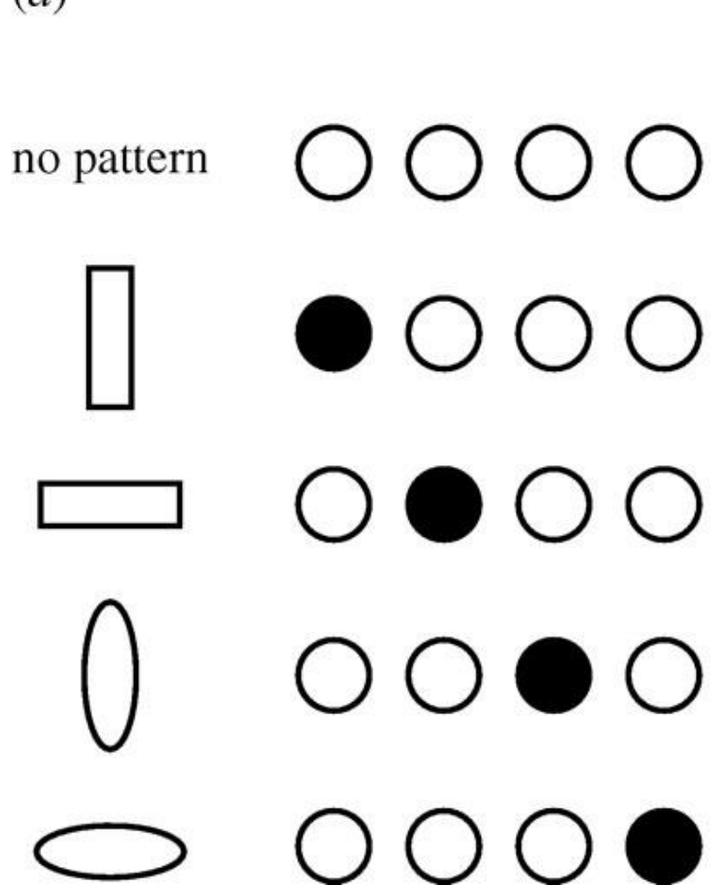
Three key ideas

- (Hierarchical) Compositionality
  - Cascade of non-linear transformations
  - Multiple layers of representations
- End-to-End Learning
  - Learning (goal-driven) representations
  - Learning to feature extract
- **Distributed Representations**
  - No single neuron “encodes” everything
  - Groups of neurons work together

slide by Dhruv Batra

Localist representations

- The simplest way to represent things with neural networks is to **dedicate one neuron to each thing**.
  - Easy to understand.
  - Easy to code by hand
    - Often used to represent inputs to a net
  - Easy to learn
    - This is what mixture models do.
    - Each cluster corresponds to one neuron
  - Easy to associate with other representations or responses.



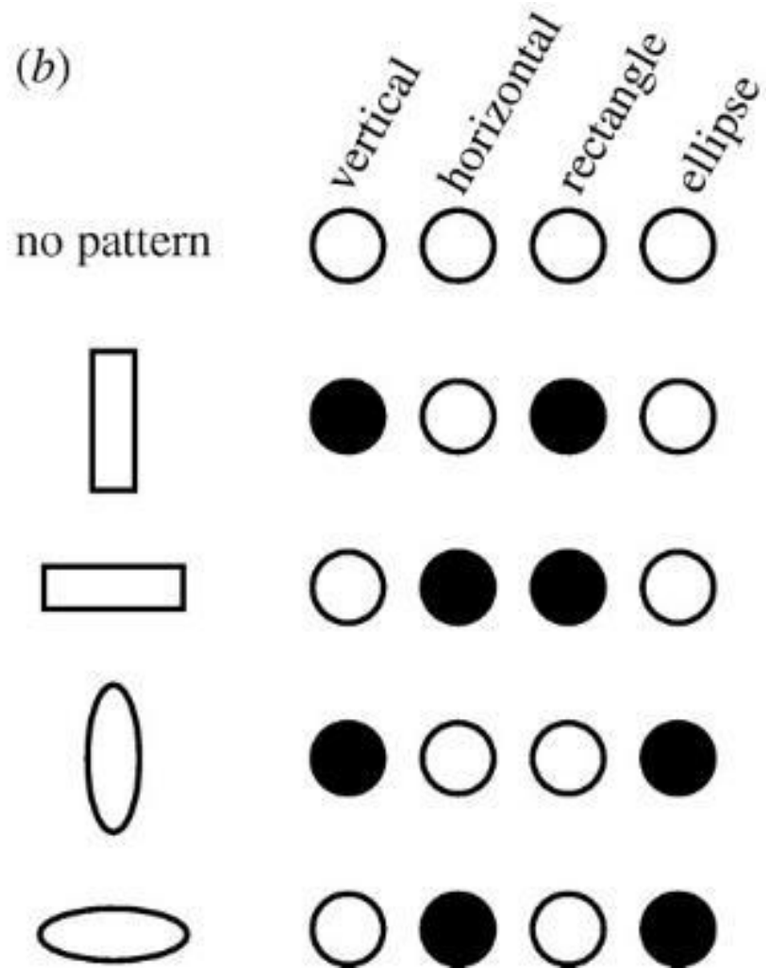
- But localist models are very inefficient whenever the data has componential structure.

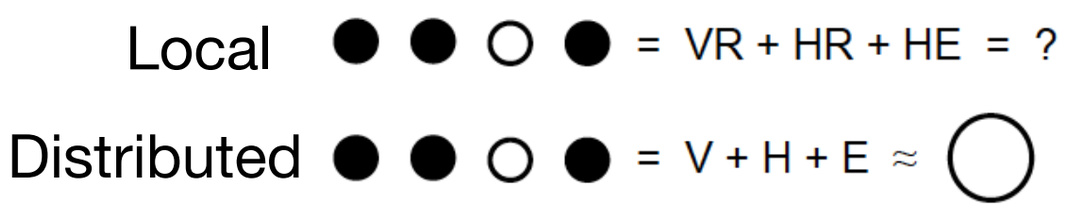
Image credit: Moontae Lee

slide by Geoff Hinton

Distributed Representations

- Each neuron must represent something, so this must be a local representation.
- **Distributed representation** means a many-to-many relationship between two types of representation (such as concepts and neurons).
  - Each concept is represented by many neurons
  - Each neuron participates in the representation of many concepts





slide by Geoff Hinton

Image credit: Moontae Lee

Power of distributed representations!

Scene Classification

bedroom

mountain

- Possible internal representations:
  - Objects
  - Scene attributes
  - Object parts
  - Textures

B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba **"Object Detectors Emerge in Deep Scene CNNs"**, ICLR 2015

slide by Bolei Zhou

Deep Convolutional Neural Networks

Convolutions

- Images typically have invariant patterns
  - E.g., directional gradients are translational invariant:

- Apply convolution to local sliding windows

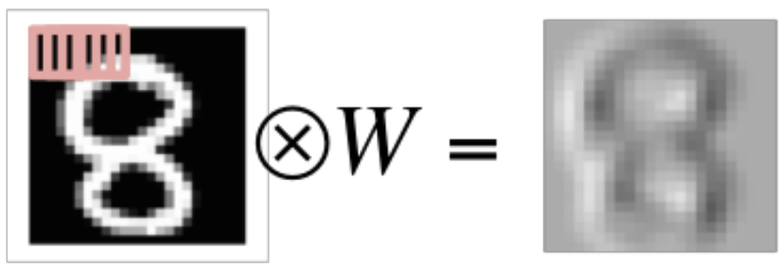
slide by Yisong Yue

Convolution Filters

- Applies to an image patch x
  - Converts local window into single value
  - Slide across image

$$x \otimes W = \sum_{ij} W_{ij} x_{ij}$$

Local Image Patch



Left-to-Right
Edge Detector

-1	0	+1
-1	0	+1
-1	0	+1

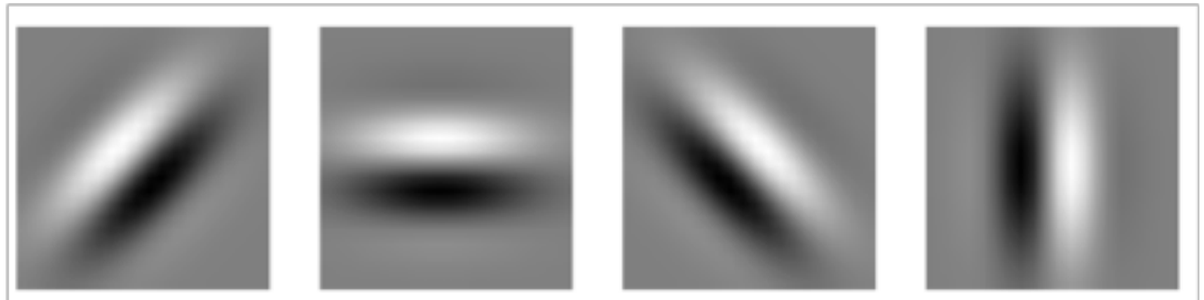
W

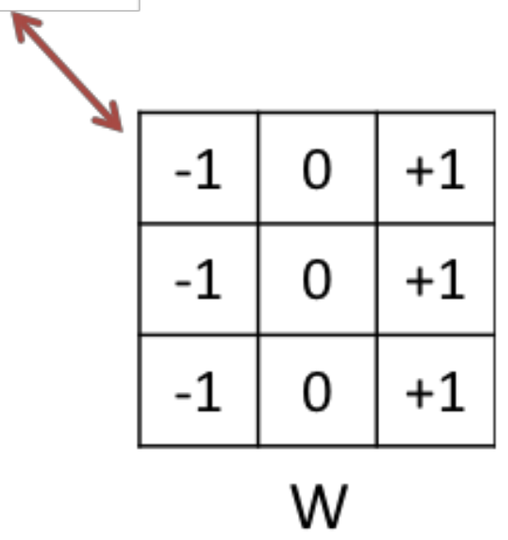
slide by Yisong Yue

Gabor Filters

- Most common low-level convolutions for computer vision

Example W:





http://en.wikipedia.org/wiki/Gabor_filter

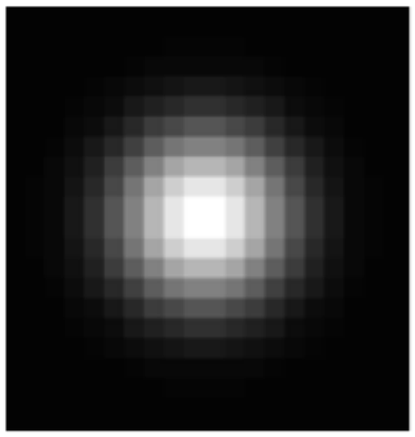
slide by Yisong Yue

Gaussian Blur Filters

- Weights decay according to Gaussian Distribution
  - Variance term controls radius

Example W:

Apply per RGB Channel



- Black = 0
- White = Positive

http://en.wikipedia.org/wiki/Gaussian_blur

slide by Yisong Yue