Lecture 1
Introduction
CMSC 35246: Deep Learning

Shubhendu Trivedi
&
Risi Kondor

University of Chicago

March 27, 2017

Administrivia

- Lectures in Ryerson 277: Monday and Wednesday 1500-1620
- Website: http://ttic.uchicago.edu/ shubhendu/Pages/CMSC35246.html; Also will use Chalk
- Additional Lab sessions if needed will be announced

Administrivia

- Lectures in Ryerson 277: Monday and Wednesday 1500-1620
- Website: http://ttic.uchicago.edu/ shubhendu/Pages/CMSC35246.html; Also will use Chalk
- Additional Lab sessions if needed will be announced
- 6 short 15 minute quizzes (no surprises, only to ensure that material is revisited)

Administrivia

- Lectures in Ryerson 277: Monday and Wednesday 1500-1620
- Website: http://ttic.uchicago.edu/ shubhendu/Pages/CMSC35246.html; Also will use Chalk
- Additional Lab sessions if needed will be announced
- 6 short 15 minute quizzes (no surprises, only to ensure that material is revisited)
- 3-4 short assignments to train networks covered in class

Administrivia

- Lectures in Ryerson 277: Monday and Wednesday 1500-1620
- Website: http://ttic.uchicago.edu/ shubhendu/Pages/CMSC35246.html; Also will use Chalk
- Additional Lab sessions if needed will be announced
- 6 short 15 minute quizzes (no surprises, only to ensure that material is revisited)
- 3-4 short assignments to train networks covered in class
- In-class midterm

Administrivia

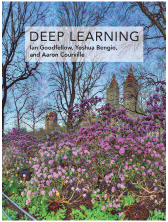
- Lectures in Ryerson 277: Monday and Wednesday 1500-1620
- Website: http://ttic.uchicago.edu/ shubhendu/Pages/CMSC35246.html; Also will use Chalk
- Additional Lab sessions if needed will be announced
- 6 short 15 minute quizzes (no surprises, only to ensure that material is revisited)
- 3-4 short assignments to train networks covered in class
- In-class midterm
- Class project in groups of 2 or alone (could be an application of Neural Networks to your own research, or on a subject suggested)

Administrivia

- Lectures in Ryerson 277: Monday and Wednesday 1500-1620
- Website: http://ttic.uchicago.edu/ shubhendu/Pages/CMSC35246.html; Also will use Chalk
- Additional Lab sessions if needed will be announced
- 6 short 15 minute quizzes (no surprises, only to ensure that material is revisited)
- 3-4 short assignments to train networks covered in class
- In-class midterm
- Class project in groups of 2 or alone (could be an application of Neural Networks to your own research, or on a subject suggested)
- Experimental course - plan subject to revision!

Books and Resources

- We will mostly follow **Deep Learning** by *Ian Goodfellow*, *Yoshua Bengio* and *Aaron Courville* (MIT Press, 2016)



- **Learning Deep Architectures for AI** by *Yoshua Bengio* (Foundations and Trends in Machine Learning, 2009)
- Additional resources:
  - **Stanford CS 231n**: by *Li, Karpathy & Johnson*
  - **Neural Networks and Deep Learning** by *Michael Nielsen*

Recommended Background

- Intro level Machine Learning:
  - STAT 37710/CMSC 35400 or TTIC 31020 or equivalent
  - CMSC 25400/STAT 27725 should be fine too!
  - Intermediate level familiarity with Maximum Likelihood Estimation, formulating cost functions, optimization with gradient descent etc. from above courses
- Good grasp of basic probability theory
- Basic Linear Algebra and Calculus
- Programming proficiency in Python (experience in some other high level language should be fine)

Contact Information

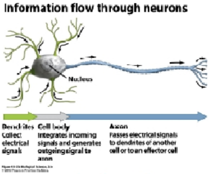
- Please fill out the questionaire linked to from the website (also on chalk)
- **Office hours**:
  - **Shubhendu Trivedi**: Mon/Wed 1630-1730, Fri 1700-1900; e-mail shubhendu@cs.uchicago.edu
  - **Risi Kondor**: TBD; e-mail risi@cs.uchicago.edu
- TA: No TA assigned (yet!)

Goals of the Course

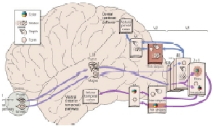
- Get a solid understanding of the nuts and bolts of Supervised Neural Networks (Feedforward, Recurrent)
- Understand selected Neural Generative Models and survey current research efforts
- A general understanding of optimization strategies to guide training Deep Architectures
- The ability to design from scratch, and train novel deep architectures
- Pick up the basics of a general purpose Neural Networks toolbox

A Brief History of Neural Networks

Neural Networks



simple functions

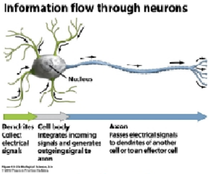


(Van Essen&Gallant, 1994)

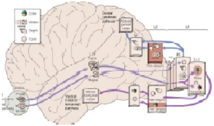
Multi-layered

- Connectionism has a long and illustrious history (could be a separate course!)

Neural Networks



simple functions

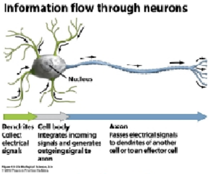


(Van Essen&Gallant, 1994)

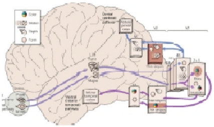
Multi-layered

- Connectionism has a long and illustrious history (could be a separate course!)
- Neurons are simple. But their arrangement in multi-layered networks is very powerful

Neural Networks



simple functions

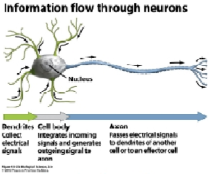


(Van Essen&Gallant, 1994)

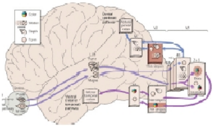
Multi-layered

- Connectionism has a long and illustrious history (could be a separate course!)
- Neurons are simple. But their arrangement in multi-layered networks is very powerful
- They self organize. Learning effectively is change in organization (or connection strengths).

Neural Networks



simple functions

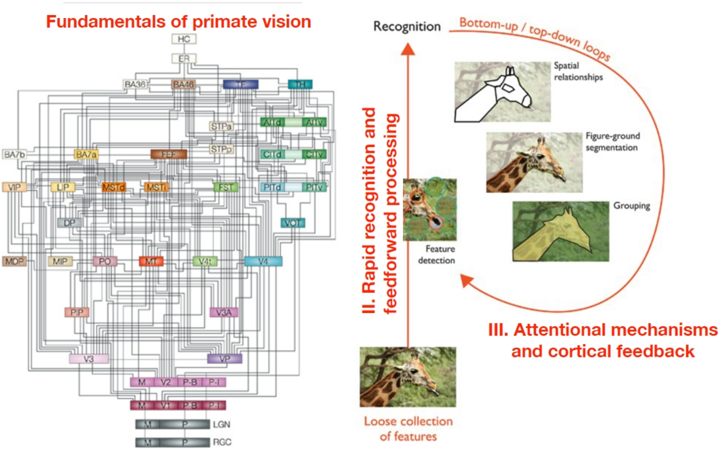


(Van Essen&Gallant, 1994)

Multi-layered

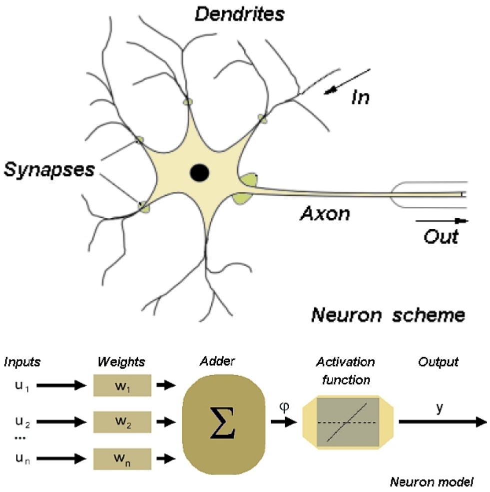
- Connectionism has a long and illustrious history (could be a separate course!)
- Neurons are simple. But their arrangement in multi-layered networks is very powerful
- They self organize. Learning effectively is change in organization (or connection strengths).
- Humans are very good at recognizing patterns. How does the brain do it?

Neural Networks



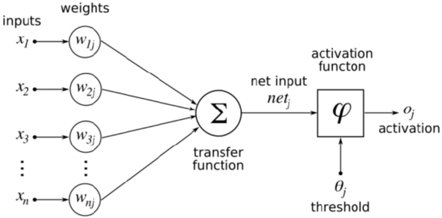
[Slide credit: Thomas Serre]

First Generation Neural Networks: McCullogh Pitts (1943)



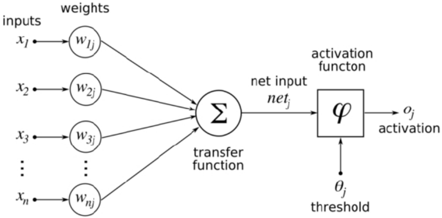
A Model Adaptive Neuron

$$o = \phi(\theta + \sum_p w_p x_p),$$



A Model Adaptive Neuron

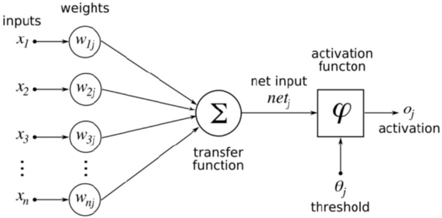
$$o = \phi(\theta + \sum_p w_p x_p),$$



- This is also called a Perceptron

A Model Adaptive Neuron

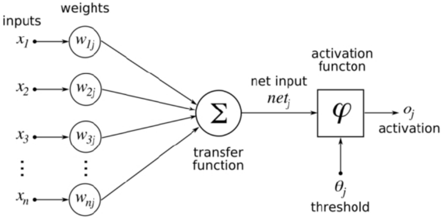
$$o = \phi(\theta + \sum_p w_p x_p),$$



- This is also called a Perceptron
- Assumes data are linearly separable. Simple stochastic algorithm for learning the linear classifier

A Model Adaptive Neuron

$$o = \phi(\theta + \sum_p w_p x_p),$$



- This is also called a Perceptron
- Assumes data are linearly separable. Simple stochastic algorithm for learning the linear classifier
- Theorem (Novikoff, 1962): Let $\mathbf{w}$, w_0 be a linear separator with $\|w\| = 1$, and margin γ. Then Perceptron will converge after

$$O\Big(\frac{(\max_i \|x_i\|)^2}{\gamma^2}\Big)$$

Algorithm

Algorithm

- Problem: Given a sequence of labeled examples $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots,$ where each $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{+1, -1\}$, find a weight vector $\mathbf{w}$ and intercept b such that $sign(\mathbf{w}\mathbf{x}_i + b) = y_i$ for all i

Algorithm

- Problem: Given a sequence of labeled examples $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots,$ where each $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{+1, -1\}$, find a weight vector $\mathbf{w}$ and intercept b such that $sign(\mathbf{w}\mathbf{x}_i + b) = y_i$ for all i
- Perceptron Algorithm

Algorithm

- Problem: Given a sequence of labeled examples $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots,$ where each $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{+1, -1\}$, find a weight vector $\mathbf{w}$ and intercept b such that $sign(\mathbf{w}\mathbf{x}_i + b) = y_i$ for all i
- Perceptron Algorithm
  - initialize $\mathbf{w} = 0$

Algorithm

- Problem: Given a sequence of labeled examples $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots,$ where each $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{+1, -1\}$, find a weight vector $\mathbf{w}$ and intercept b such that $sign(\mathbf{w}\mathbf{x}_i + b) = y_i$ for all i
- Perceptron Algorithm
  - initialize $\mathbf{w} = 0$
  - if $sign(\mathbf{w}\mathbf{x}) \neq y$ (mistake), then $\mathbf{w}_{new} = \mathbf{w}_{old} + \eta y \mathbf{x}$ (η is learning rate)

Perceptron as a model of the brain?

- Perceptron developed in the 1950s

Perceptron as a model of the brain?

- Perceptron developed in the 1950s
- Key publication: *The perceptron: a probabilistic model for information storage and organization in the brain*, Frank Rosenblatt, Psychological Review, 1958

Perceptron as a model of the brain?

- Perceptron developed in the 1950s
- Key publication: *The perceptron: a probabilistic model for information storage and organization in the brain*, Frank Rosenblatt, Psychological Review, 1958
- Goal: Pattern classification

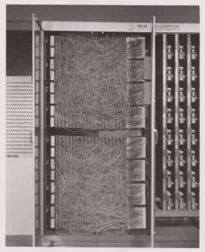
Perceptron as a model of the brain?

- Perceptron developed in the 1950s
- Key publication: *The perceptron: a probabilistic model for information storage and organization in the brain*, Frank Rosenblatt, Psychological Review, 1958
- Goal: Pattern classification
- From Mechanization of Thought Process (1959): "The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence. Later perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech and writing in another language, it was predicted."

Perceptron as a model of the brain?

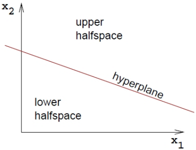
- Perceptron developed in the 1950s
- Key publication: *The perceptron: a probabilistic model for information storage and organization in the brain*, Frank Rosenblatt, Psychological Review, 1958
- Goal: Pattern classification
- From Mechanization of Thought Process (1959): "The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence. Later perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech and writing in another language, it was predicted."
- Another ancient milestone: Hebbian learning rule (Donald Hebb, 1949)

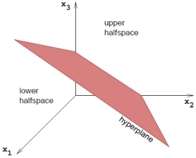
Perceptron as a model of the brain?



- The Mark I perceptron machine was the first implementation of the perceptron algorithm.
- The machine was connected to a camera that used 2020 cadmium sulfide photocells to produce a 400-pixel image. The main visible feature is a patchboard that allowed experimentation with different combinations of input features.
- To the right of that are arrays of potentiometers that implemented the adaptive weights

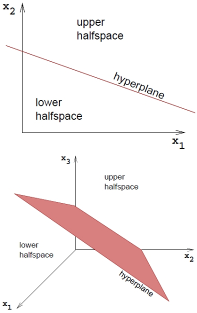
Adaptive Neuron: Perceptron





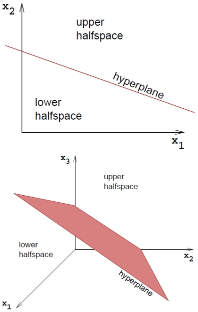
- A perceptron represents a decision surface in a d dimensional space as a hyperplane

Adaptive Neuron: Perceptron



- A perceptron represents a decision surface in a d dimensional space as a hyperplane
- Works only for those sets of examples that are *linearly separable*

Adaptive Neuron: Perceptron



- A perceptron represents a decision surface in a d dimensional space as a hyperplane
- Works only for those sets of examples that are *linearly separable*
- Many boolean functions can be represented by a perceptron: AND, OR, NAND,NOR

Problems?

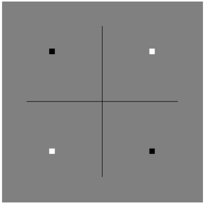
- If features are complex enough, anything can be classified

Problems?

- If features are complex enough, anything can be classified
- Thus features are really hand coded. But it comes with a clever algorithm for weight updates

Problems?

- If features are complex enough, anything can be classified
- Thus features are really hand coded. But it comes with a clever algorithm for weight updates
- If features are restricted, then some interesting tasks cannot be learned and thus perceptrons are fundamentally limited in what they can do. Famous examples: XOR, Group Invariance Theorems (Minsky, Papert, 1969)



Coda

- Single neurons are not able to solve complex tasks (linear decision boundaries).

Coda

- Single neurons are not able to solve complex tasks (linear decision boundaries).
- More layers of linear units are not enough (still linear).

Coda

- Single neurons are not able to solve complex tasks (linear decision boundaries).
- More layers of linear units are not enough (still linear).
- We could have multiple layers of adaptive, non-linear hidden units. These are called Multi-layer perceptrons

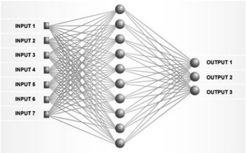
Coda

- Single neurons are not able to solve complex tasks (linear decision boundaries).
- More layers of linear units are not enough (still linear).
- We could have multiple layers of adaptive, non-linear hidden units. These are called Multi-layer perceptrons
- Many local minima: Perceptron convergence theorem does not apply.

Coda

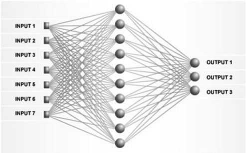
- Single neurons are not able to solve complex tasks (linear decision boundaries).
- More layers of linear units are not enough (still linear).
- We could have multiple layers of adaptive, non-linear hidden units. These are called Multi-layer perceptrons
- Many local minima: Perceptron convergence theorem does not apply.
- Intuitive conjecture in the 60s: There is no learning algorithm for multilayer perceptrons

Second Wave: Multi-layer Perceptrons



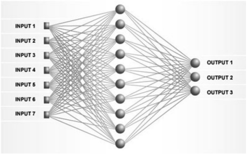
- We have looked at how each neuron will look like

Second Wave: Multi-layer Perceptrons

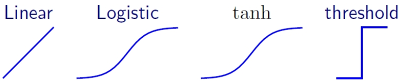


- We have looked at how each neuron will look like
- But did not mention activation functions. Some common choices:

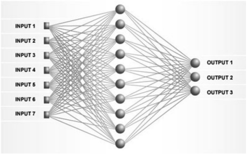
Second Wave: Multi-layer Perceptrons



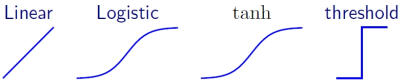
- We have looked at how each neuron will look like
- But did not mention activation functions. Some common choices:



Second Wave: Multi-layer Perceptrons

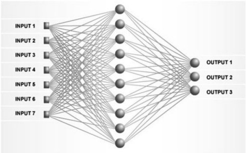


- We have looked at how each neuron will look like
- But did not mention activation functions. Some common choices:

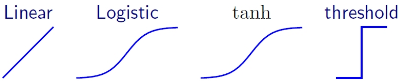


- How can we learn the weights?

Second Wave: Multi-layer Perceptrons

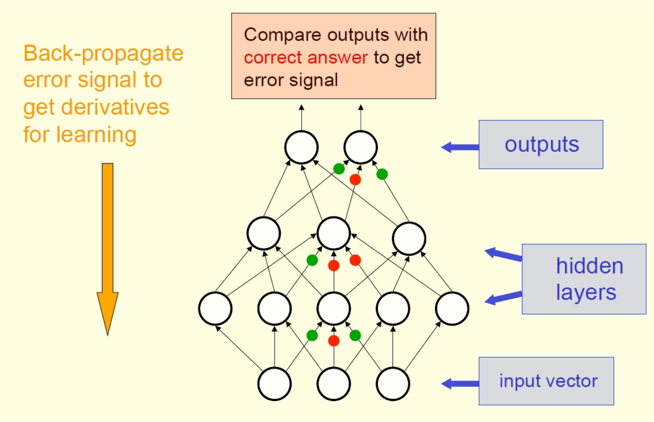


- We have looked at how each neuron will look like
- But did not mention activation functions. Some common choices:



- How can we learn the weights?
- PS: There were many kinds of Neural Models explored in the second wave (will see later in the course)

Learning multiple layers of features



[Slide: G. E. Hinton]

Multilayer Perceptrons

- Theoretical result [Cybenko, 1989]: 2-layer net with linear output can approximate any continuous function over compact domain to arbitrary accuracy (given enough hidden units!)

Multilayer Perceptrons

- Theoretical result [Cybenko, 1989]: 2-layer net with linear output can approximate any continuous function over compact domain to arbitrary accuracy (given enough hidden units!)
- The more number of hidden layers, the better...

Multilayer Perceptrons

- Theoretical result [Cybenko, 1989]: 2-layer net with linear output can approximate any continuous function over compact domain to arbitrary accuracy (given enough hidden units!)
- The more number of hidden layers, the better...
- .. in theory.

Multilayer Perceptrons

- Theoretical result [Cybenko, 1989]: 2-layer net with linear output can approximate any continuous function over compact domain to arbitrary accuracy (given enough hidden units!)
- The more number of hidden layers, the better...
- .. in theory.
- In practice deeper neural networks would need a lot of labeled data and could be not trained easily

Multilayer Perceptrons

- Theoretical result [Cybenko, 1989]: 2-layer net with linear output can approximate any continuous function over compact domain to arbitrary accuracy (given enough hidden units!)
- The more number of hidden layers, the better...
- .. in theory.
- In practice deeper neural networks would need a lot of labeled data and could be not trained easily
- Neural Networks and Backpropagation (with the exception of use in Convolutional Networks) went out of fashion between 1990-2006

Multilayer Perceptrons

- Theoretical result [Cybenko, 1989]: 2-layer net with linear output can approximate any continuous function over compact domain to arbitrary accuracy (given enough hidden units!)
- The more number of hidden layers, the better...
- .. in theory.
- In practice deeper neural networks would need a lot of labeled data and could be not trained easily
- Neural Networks and Backpropagation (with the exception of use in Convolutional Networks) went out of fashion between 1990-2006
- Digression: Kernel Methods

Return

- In 2006 Hinton and colleagues found a way to pre-train feedforward networks using a Deep Belief Network trained greedily

Return

- In 2006 Hinton and colleagues found a way to pre-train feedforward networks using a Deep Belief Network trained greedily
- This allowed larger networks to be trained by simply using backpropagation for fine tuning the pre-trained network (easier!)

Return

- In 2006 Hinton and colleagues found a way to pre-train feedforward networks using a Deep Belief Network trained greedily
- This allowed larger networks to be trained by simply using backpropagation for fine tuning the pre-trained network (easier!)
- Since 2010 pre-training of large feedforward networks in this sense also out

Return

- In 2006 Hinton and colleagues found a way to pre-train feedforward networks using a Deep Belief Network trained greedily
- This allowed larger networks to be trained by simply using backpropagation for fine tuning the pre-trained network (easier!)
- Since 2010 pre-training of large feedforward networks in this sense also out
- Availability of large datasets and fast GPU implementations have made backpropagation from scratch almost unbeatable

Return

- In 2006 Hinton and colleagues found a way to pre-train feedforward networks using a Deep Belief Network trained greedily
- This allowed larger networks to be trained by simply using backpropagation for fine tuning the pre-trained network (easier!)
- Since 2010 pre-training of large feedforward networks in this sense also out
- Availability of large datasets and fast GPU implementations have made backpropagation from scratch almost unbeatable

Why use Deep Multi Layered Models?

Argument 1: Visual scenes are hierarchially organized (so is language!)

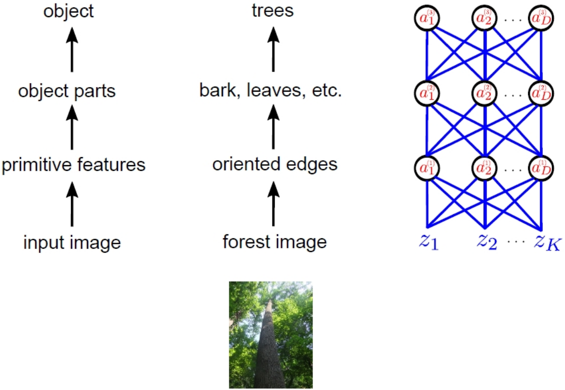


Figure: Richard E. Turner

Why use Deep Multi Layered Models?

Argument 2: Biological vision is hierarchically organized, and we want to glean some ideas from there

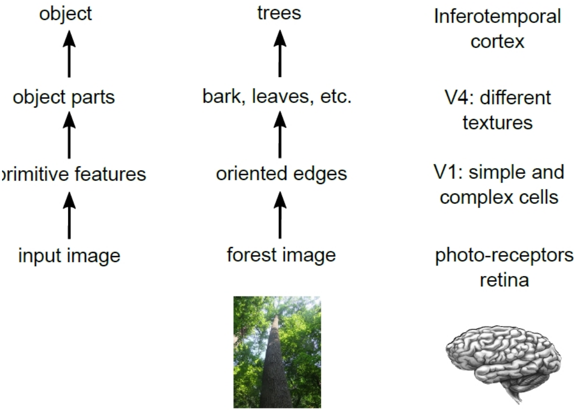
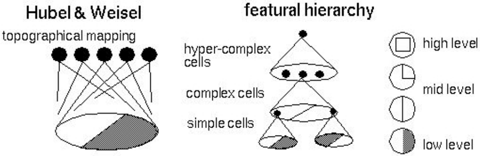


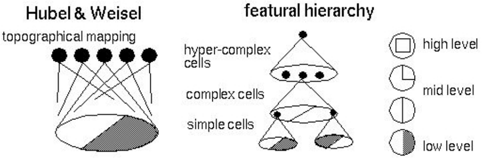
Figure: Richard E. Turner

- In the perceptual system, neurons represent features of the sensory input

- In the perceptual system, neurons represent features of the sensory input
- The brain learns to extract many layers of features. Features in one layer represent more complex combinations of features in the layer below. (e.g. Hubel Weisel (Vision), 1959, 1962)



- In the perceptual system, neurons represent features of the sensory input
- The brain learns to extract many layers of features. Features in one layer represent more complex combinations of features in the layer below. (e.g. Hubel Weisel (Vision), 1959, 1962)



- How can we imitate such a process on a computer?

Why use Deep Multi Layered Models?

Argument 3: Shallow representations are inefficient at representing highly varying functions

Why use Deep Multi Layered Models?

Argument 3: Shallow representations are inefficient at representing highly varying functions

- *when a function can be compactly represented by a deep architecture, it might need a very large architecture to be represented by an insufficiently deep one*

Why use Deep Multi Layered Models?

Argument 3: Shallow representations are inefficient at representing highly varying functions

- *when a function can be compactly represented by a deep architecture, it might need a very large architecture to be represented by an insufficiently deep one*
- Is there a theoretical justification?

Why use Deep Multi Layered Models?

Argument 3: Shallow representations are inefficient at representing highly varying functions

- *when a function can be compactly represented by a deep architecture, it might need a very large architecture to be represented by an insufficiently deep one*
- Is there a theoretical justification? No

Why use Deep Multi Layered Models?

Argument 3: Shallow representations are inefficient at representing highly varying functions

- *when a function can be compactly represented by a deep architecture, it might need a very large architecture to be represented by an insufficiently deep one*
- Is there a theoretical justification? No
- Suggestive results:

Why use Deep Multi Layered Models?

Argument 3: Shallow representations are inefficient at representing highly varying functions

Why use Deep Multi Layered Models?

Argument 3: Shallow representations are inefficient at representing highly varying functions

- A two layer circuit of logic gates can represent any Boolean function (Mendelson, 1997)

Why use Deep Multi Layered Models?

Argument 3: Shallow representations are inefficient at representing highly varying functions

- A two layer circuit of logic gates can represent any Boolean function (Mendelson, 1997)
- First result: With depth-two logical circuits, most Boolean functions need an exponential number of logic gates

Why use Deep Multi Layered Models?

Argument 3: Shallow representations are inefficient at representing highly varying functions

- A two layer circuit of logic gates can represent any Boolean function (Mendelson, 1997)
- First result: With depth-two logical circuits, most Boolean functions need an exponential number of logic gates
- Another result (Hastad, 1986): There exist functions with poly-size logic gate circuit of depth k that require exponential size when restricted to depth $k-1$

Why use Deep Multi Layered Models?

Argument 3: Shallow representations are inefficient at representing highly varying functions

- A two layer circuit of logic gates can represent any Boolean function (Mendelson, 1997)
- First result: With depth-two logical circuits, most Boolean functions need an exponential number of logic gates
- Another result (Hastad, 1986): There exist functions with poly-size logic gate circuit of depth k that require exponential size when restricted to depth $k-1$
- Why do we care about boolean circuits?

Why use Deep Multi Layered Models?

Argument 3: Shallow representations are inefficient at representing highly varying functions

- A two layer circuit of logic gates can represent any Boolean function (Mendelson, 1997)
- First result: With depth-two logical circuits, most Boolean functions need an exponential number of logic gates
- Another result (Hastad, 1986): There exist functions with poly-size logic gate circuit of depth k that require exponential size when restricted to depth $k-1$
- Why do we care about boolean circuits?
- Similar results are known when the computational units are linear threshold units (Hastad, Razborov)

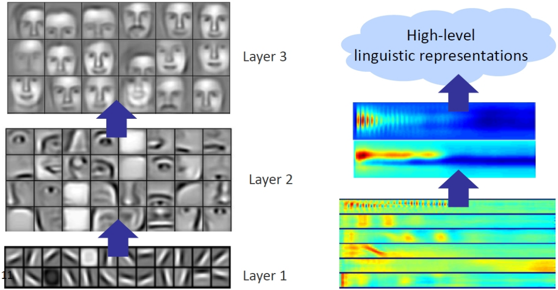
Why use Deep Multi Layered Models?

Argument 3: Shallow representations are inefficient at representing highly varying functions

- A two layer circuit of logic gates can represent any Boolean function (Mendelson, 1997)
- First result: With depth-two logical circuits, most Boolean functions need an exponential number of logic gates
- Another result (Hastad, 1986): There exist functions with poly-size logic gate circuit of depth k that require exponential size when restricted to depth $k - 1$
- Why do we care about boolean circuits?
- Similar results are known when the computational units are linear threshold units (Hastad, Razborov)
- In practice depth helps in complicated tasks

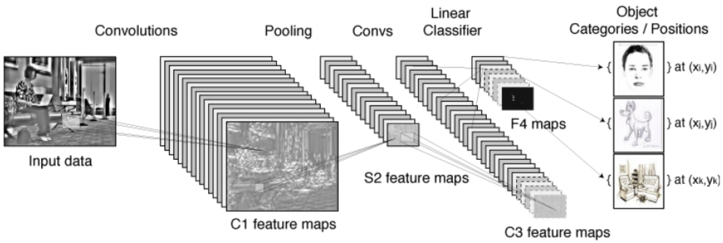
Why use Deep Multi Layered Models?

- Attempt to learn features and the entire pipeline end-to-end rather than engineering it (the engineering focus shifts to architecture design)



[Figure: Honglak Lee]

Convolutional Neural Networks



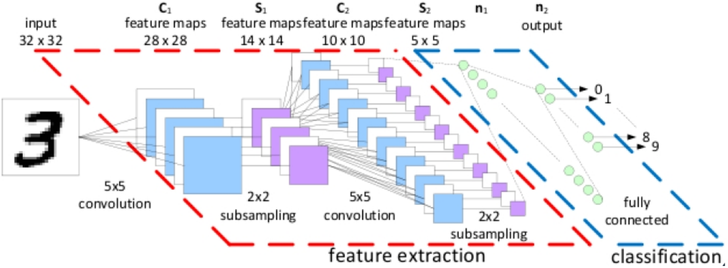


Figure: Yann LeCun

Convolutional Neural Networks

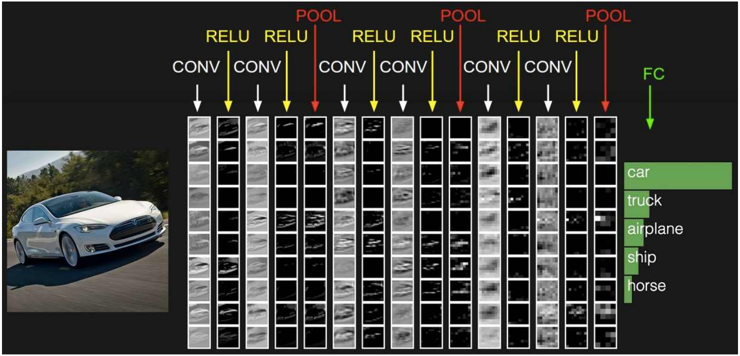
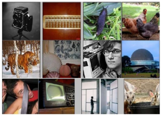


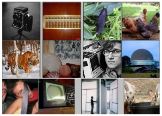
Figure: Andrej Karpathy

ImageNet Challenge 2012



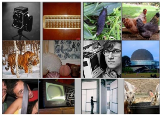
- 14 million labeled images with 20,000 classes

ImageNet Challenge 2012



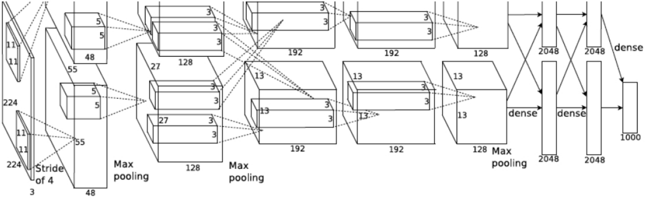
- 14 million labeled images with 20,000 classes
- Images gathered from the internet and labeled by humans via Amazon Turk

ImageNet Challenge 2012



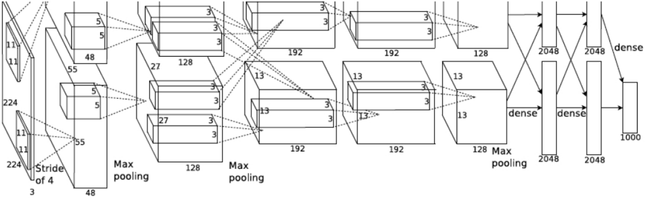
- 14 million labeled images with 20,000 classes
- Images gathered from the internet and labeled by humans via Amazon Turk
- Challenge: 1.2 million training images, 1000 classes.

ImageNet Challenge 2012



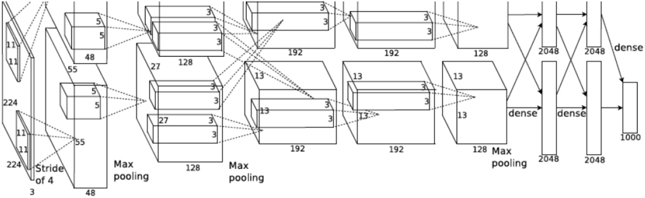
- Winning model ("AlexNet") was a convolutional network similar to Yann LeCun, 1998

ImageNet Challenge 2012



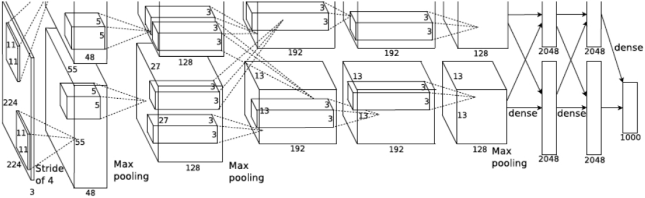
- Winning model ("AlexNet") was a convolutional network similar to Yann LeCun, 1998
- More data: 1.2 million versus a few thousand images

ImageNet Challenge 2012



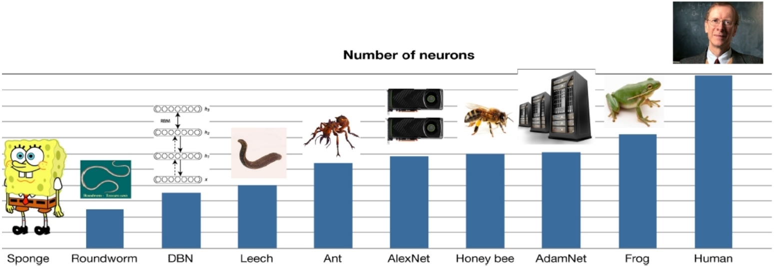
- Winning model ("AlexNet") was a convolutional network similar to Yann LeCun, 1998
- More data: 1.2 million versus a few thousand images
- Fast two GPU implementation trained for a week

ImageNet Challenge 2012



- Winning model ("AlexNet") was a convolutional network similar to Yann LeCun, 1998
- More data: 1.2 million versus a few thousand images
- Fast two GPU implementation trained for a week
- Better regularization

[A. Krizhevsky, I. Sutskever, G. E. Hinton: ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012]



(Goodfellow, 2013)

Going Deeper

- A lot of current research has focussed on architecture (efficient, deeper, faster to train)

Going Deeper

- A lot of current research has focussed on architecture (efficient, deeper, faster to train)
- Examples: VGGNet, Inception, Highway Networks, Residual Networks, Fractal Networks

Going Deeper

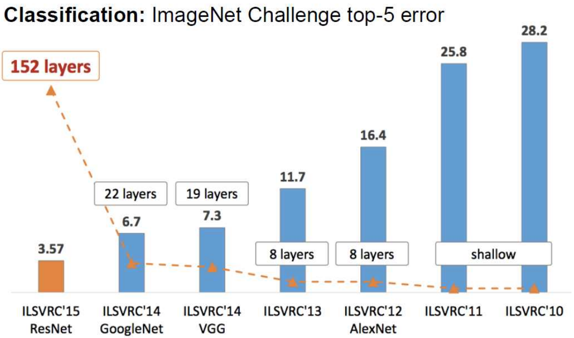
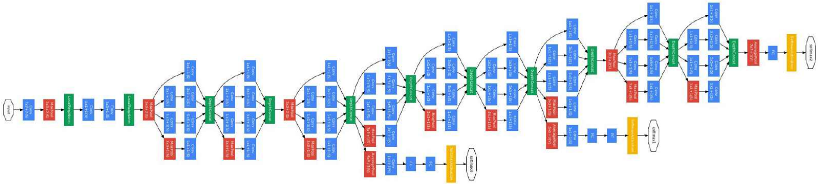


Figure: Kaiming He, MSR

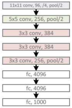
Google LeNet



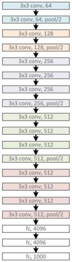
C. Szegedy et al, Going Deeper With Convolutions, CVPR 2015

Revolution of Depth

AlexNet, 8 layers
(ILSVRC 2012)



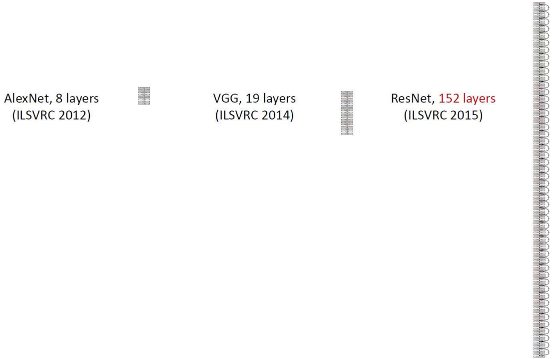
VGG, 19 layers
(ILSVRC 2014)



GoogleNet, 22 layers
(ILSVRC 2014)

K. He et al, Deep Residual Learning for Image Recognition, CVPR 2016. Slide: K. He

Revolution of Depth



K. He et al, Deep Residual Learning for Image Recognition, CVPR 2016. Slide: K. He

Residual Networks

- Number 1 in Image classification
- ImageNet Detection: 16 % better than the second best
- ImageNet Localization: 27 % better than the second best
- COCO Detection: 11 % better than the second best
- COCO Segmentation: 12 % better than the second best

Sequence Tasks

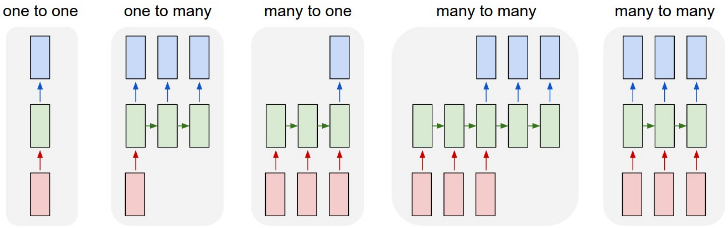


Figure credit: Andrej Karpathy

Recent Deep Learning Successes and Research Areas

2016: Year of Deep Learning

SHARE

SHARE 1854

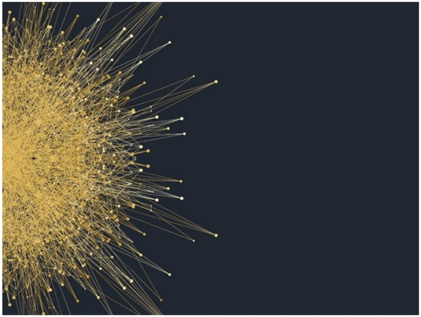
TWEET

COMMENT 1

EMAIL

CADE METZ BUSINESS 12.25.16 7:00 AM

2016: THE YEAR THAT DEEP LEARNING TOOK OVER THE INTERNET



Even Star Power! :)

Artificial Intelligence cybernetics machine learning technology film

Kristen Stewart co-authored a paper on style transfer and the AI community lost its mind

Posted Jan 19, 2017 by John Mannes (@JohnMannes)

Next Story

Maybe Hyped?

MIT Technology Review

Log in / Register Search

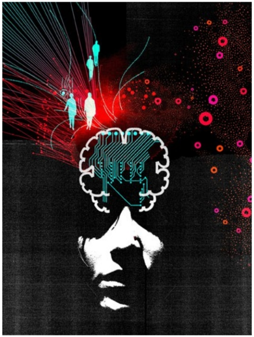
Past Lists+ Topics+ Top Stories Magazine Events More+ Subscribe

10 Breakthrough Technologies The List+ Years+

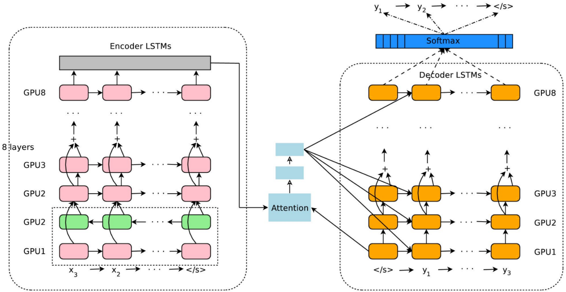
Deep Learning

With massive amounts of computational power, machines can now recognize objects and translate speech in real time. Artificial intelligence is finally getting smart.

by Robert D. Hof



Machine Translation



- Your Google Translate usage will now be powered by an 8 layer Long Short Term Memory Network with residual connections and attention

Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation; Wu *et al.*

Artistic Style

(a) With conditional instance normalization, a single style transfer network can capture 32 styles at the same time, five of which are shown here. All 32 styles in this single model are in the Appendix. Golden Gate Bridge photograph by Rich Niewiroski Jr.

A Learned Representation for Artistic Style; Dumoulin, Shlens, Kudlur; ICLR 2017

Speech Synthesis

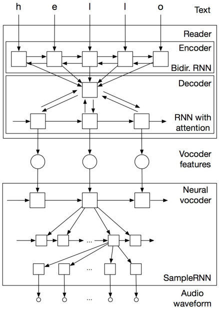


Figure 1: Char2Wav: An end-to-end speech synthesis model.

Char2Wav: End-to-End Speech Synthesis; Sotelo *et al.*, ICLR 2017; http://josesotelo.com/speechsynthesis/

Game Playing

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

Neuroevolution of Architectures

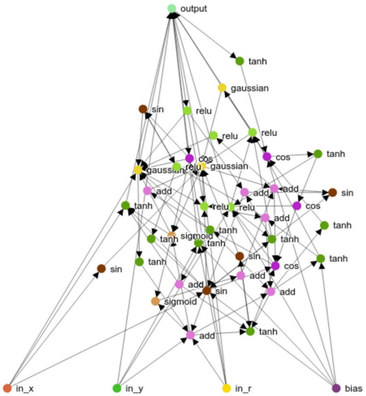


Figure: @hardmaru

- Recent large scale studies by Google show that evolutionary methods are catching with intelligently designed architectures

As well as in:

- Protein Folding

As well as in:

- Protein Folding
- Drug discovery

As well as in:

- Protein Folding
- Drug discovery
- Particle Physics

As well as in:

- Protein Folding
- Drug discovery
- Particle Physics
- Energy Management

As well as in:

- Protein Folding
- Drug discovery
- Particle Physics
- Energy Management
- ...

Next time

- Feedforward Networks

Next time

- Feedforward Networks
- Backpropagation