Foundations of Artificial Intelligence

14. Deep Learning

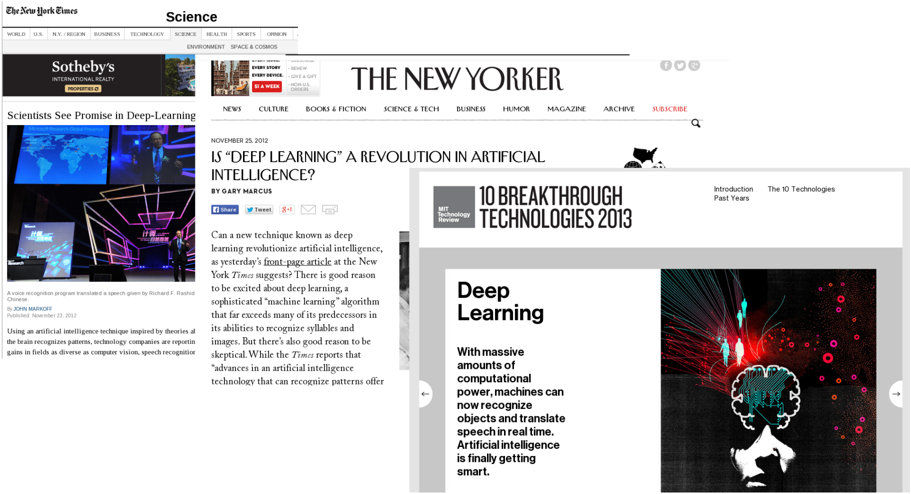
An Overview

Joschka Boedecker and Wolfram Burgard and
Frank Hutter and Bernhard Nebel

Albert-Ludwigs-Universität Freiburg

July 11, 2018

Motivation: Deep Learning in the News



The New York Times

Science

WORLD U.S. N.Y. / REGION BUSINESS TECHNOLOGY SCIENCE HEALTH SPORTS OPINION

ENVIRONMENT SPACE & COSMOS

Scientists See Promise in Deep-Learning

A voice recognition program translated a speech given by Richard F. Rashid Chinese.

By JOHN MARKOFF

Published: November 23, 2012

Using an artificial intelligence technique inspired by theories al the brain recognizes patterns, technology companies are reportin gains in fields as diverse as computer vision, speech recognitior

THE NEW YORKER

NEWS CULTURE BOOKS & FICTION SCIENCE & TECH BUSINESS HUMOR MAGAZINE ARCHIVE SUBSCRIBE

NOVEMBER 25, 2012

IS "DEEP LEARNING" A REVOLUTION IN ARTIFICIAL INTELLIGENCE?

BY GARY MARCUS

Can a new technique known as deep learning revolutionize artificial intelligence, as yesterday's front-page article at the New York *Times* suggests? There is good reason to be excited about deep learning, a sophisticated "machine learning" algorithm that far exceeds many of its predecessors in its abilities to recognize syllables and images. But there's also good reason to be skeptical. While the *Times* reports that "advances in an artificial intelligence technology that can recognize patterns offer

MIT Technology Review

10 BREAKTHROUGH TECHNOLOGIES 2013

Introduction The 10 Technologies

Past Years

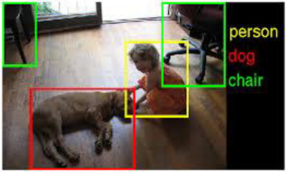
Deep Learning

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

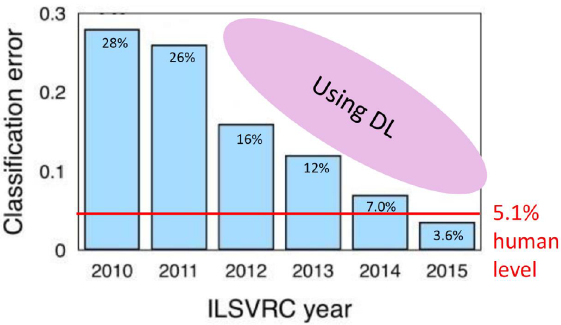
Motivation: Why is Deep Learning so Popular?

- Excellent empirical results, e.g., in computer vision

Object recognition



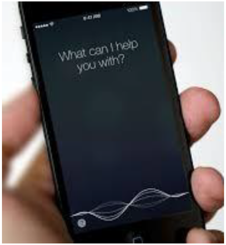
Self-driving cars



Motivation: Why is Deep Learning so Popular?

- Excellent empirical results, e.g., in speech recognition

Speech recognition



skype™

Auto-Translator

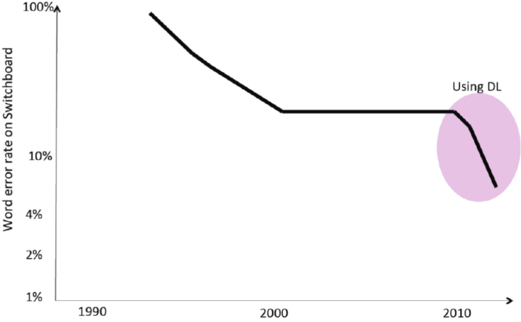
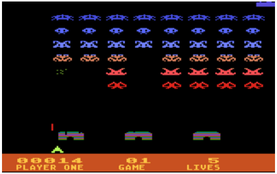


Image credit: Yoshua Bengio (data from Microsoft speech group)

Motivation: Why is Deep Learning so Popular?

- Excellent empirical results, e.g., in reasoning in games

  - Superhuman performance in playing Atari games
    [Mnih et al, Nature 2015]

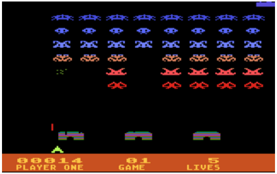


  - Beating the world's best Go player
    [Silver et al, Nature 2016]

Motivation: Why is Deep Learning so Popular?

- Excellent empirical results, e.g., in reasoning in games

    - Superhuman performance in playing Atari games [Mnih et al, Nature 2015]



    - Beating the world's best Go player [Silver et al, Nature 2016]

- More reasons for the popularity of deep learning throughout

Lecture Overview

Lecture Overview

Some definitions

Representation learning

"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"

Some definitions

Representation learning

"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"

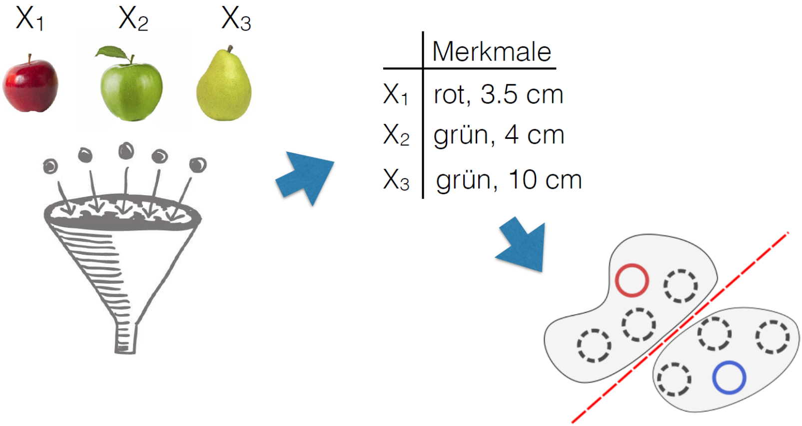
Deep learning

"representation learning methods with multiple levels of representation, obtained by composing simple but nonlinear modules that each transform the representation at one level into a [...] higher, slightly more abstract (one)"

(LeCun et al., 2015)

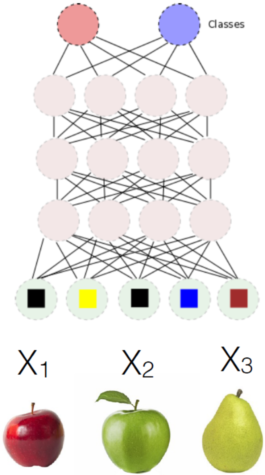
Standard Machine Learning Pipeline

- Standard machine learning algorithms are based on high-level attributes or features of the data
- E.g., the binary attributes we used for decisions trees
- This requires (often substantial) feature engineering

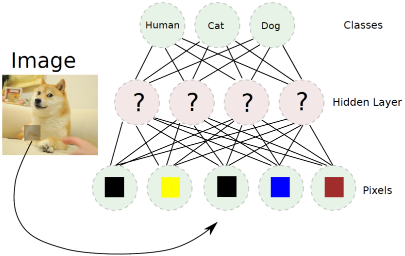


Representation Learning Pipeline

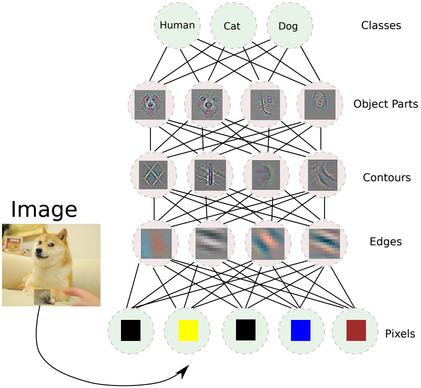
- Jointly learn features and classifier, directly from raw data
- This is also referrred to as end-to-end learning



Shallow vs. Deep Learning

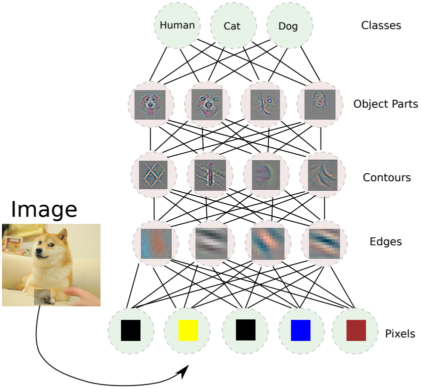


Shallow vs. Deep Learning



- Deep Learning: learning a hierarchy of representations that build on each other, from simple to complex

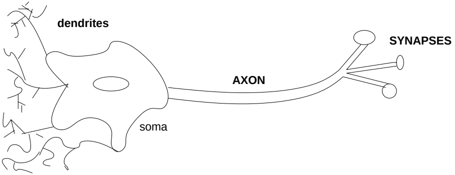
Shallow vs. Deep Learning



- Deep Learning: learning a hierarchy of representations that build on each other, from simple to complex
- Quintessential deep learning model: Multilayer Perceptrons

Biological Inspiration of Artificial Neural Networks

- Dendrites input information to the cell
- Neuron fires (has action potential) if a certain threshold for the voltage is exceeded
- Output of information by axon
- The axon is connected to dentrites of other cells via synapses
- Learning: adaptation of the synapse's efficiency, its synaptical weight



A Very Brief History of Neural Networks

- Neural networks have a long history
  - 1942: artificial neurons (McCulloch/Pitts)
  - 1958/1969: perceptron (Rosenblatt; Minsky/Papert)
  - 1986: multilayer perceptrons and backpropagation (Rumelhart)
  - 1989: convolutional neural networks (LeCun)

A Very Brief History of Neural Networks

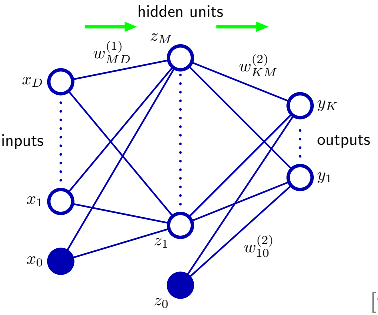
- Neural networks have a long history
  - 1942: artificial neurons (McCulloch/Pitts)
  - 1958/1969: perceptron (Rosenblatt; Minsky/Papert)
  - 1986: multilayer perceptrons and backpropagation (Rumelhart)
  - 1989: convolutional neural networks (LeCun)
- Alternative theoretically motivated methods outperformed NNs
  - Exaggerated expectations: "It works like the brain" (No, it does not!)

A Very Brief History of Neural Networks

- Neural networks have a long history
  - 1942: artificial neurons (McCulloch/Pitts)
  - 1958/1969: perceptron (Rosenblatt; Minsky/Papert)
  - 1986: multilayer perceptrons and backpropagation (Rumelhart)
  - 1989: convolutional neural networks (LeCun)
- Alternative theoretically motivated methods outperformed NNs
  - Exaggeraged expectations: "It works like the brain" (No, it does not!)
- Why the sudden success of neural networks in the last 5 years?
  - Data: Availability of massive amounts of labelled data
  - Compute power: Ability to train very large neural networks on GPUs
  - Methodological advances: many since first renewed popularization

Lecture Overview

Multilayer Perceptrons



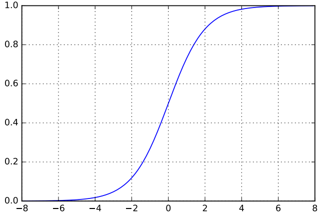
[figure from Bishop, Ch. 5]

- Network is organized in layers
  - Outputs of k-th layer serve as inputs of $k+1$th layer
- Each layer k only does quite simple computations:
  - Linear function of previous layer's outputs $\mathbf{z}_{k-1}$: $\mathbf{a}_k = \mathbf{W}_k\mathbf{z}_{k-1} + \mathbf{b}_k$
  - Nonlinear transformation $\mathbf{z}_k = h_k(\mathbf{a}_k)$ through activation function h_k

Activation Functions - Examples

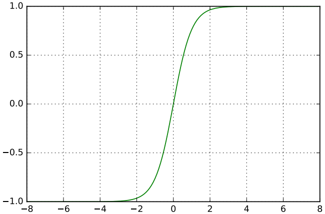
Logistic sigmoid activation function:

$$h_{logistic}(a) = \frac{1}{1 + \exp(-a)}$$



Logistic hyperbolic tangent activation function:

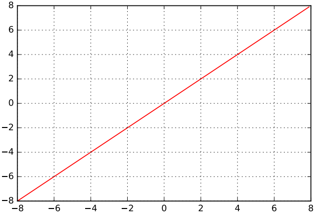
$$h_{tanh}(a) = \tanh(a)$$
$$= \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)}\}$$



Activation Functions - Examples (cont.)

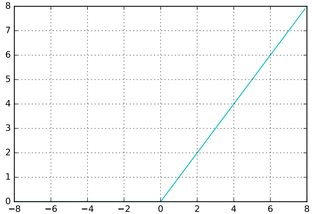
Linear activation function:

$$h_{linear}(a) = a$$



Rectified Linear (ReLU) activation function:

$$h_{relu}(a) = \max(0, a)$$



Output unit activation functions

Depending on the task, typically:

- for regression: single output neuron with linear activation

Output unit activation functions

Depending on the task, typically:

- for regression: single output neuron with linear activation
- for binary classification: single output neuron with logistic/tanh activation

Output unit activation functions

Depending on the task, typically:

- for regression: single output neuron with linear activation
- for binary classification: single output neuron with logistic/tanh activation
- for multiclass classification: K output neurons and softmax activation

$$(\hat{\mathbf{y}}(\mathbf{x}, \mathbf{w}))_k = h_{softmax}((\mathbf{a})_k) = \frac{\exp((\mathbf{a})_k)}{\sum_j \exp((\mathbf{a})_j)}$$

Output unit activation functions

Depending on the task, typically:

- for regression: single output neuron with linear activation
- for binary classification: single output neuron with logistic/tanh activation
- for multiclass classification: K output neurons and softmax activation

$$(\hat{\mathbf{y}}(\mathbf{x}, \mathbf{w}))_k = h_{softmax}((\mathbf{a})_k) = \frac{\exp((\mathbf{a})_k)}{\sum_j \exp((\mathbf{a})_j)}$$

$\rightarrow$ so for the complete output layer:

$$\hat{\mathbf{y}}(\mathbf{x}, \mathbf{w}) = \begin{bmatrix} p(y_1 = 1|\mathbf{x}) \\ p(y_2 = 1|\mathbf{x}) \\ \vdots \\ p(y_K = 1|\mathbf{x}) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp((\mathbf{a})_j)} \exp(\mathbf{a})$$

Loss function to be minimized

- Consider binary classification task using a single output unit with logistic sigmoid activation function:

$$\hat{y}(\mathbf{x}, \mathbf{w}) = h_{logistic}(a) = \frac{1}{1 + \exp(-a)}$$

Loss function to be minimized

- Consider binary classification task using a single output unit with logistic sigmoid activation function:

$$\hat{y}(\mathbf{x}, \mathbf{w}) = h_{logistic}(a) = \frac{1}{1 + \exp(-a)}$$

- This defines a (Bernoulli) probability distribution over the label of each data point $\mathbf{x}_n$:

$$\begin{aligned} p(y_n = 1 \mid \mathbf{x}_n, \mathbf{w}) &= \hat{y}(\mathbf{x}_n, \mathbf{w}) \\ p(y_n = 0 \mid \mathbf{x}_n, \mathbf{w}) &= 1 - \hat{y}(\mathbf{x}_n, \mathbf{w}) \end{aligned}$$

Loss function to be minimized

- Consider binary classification task using a single output unit with logistic sigmoid activation function:

$$\hat{y}(\mathbf{x}, \mathbf{w}) = h_{logistic}(a) = \frac{1}{1 + \exp(-a)}$$

- This defines a (Bernoulli) probability distribution over the label of each data point $\mathbf{x}_n$:

$$\begin{aligned} p(y_n = 1 \mid \mathbf{x}_n, \mathbf{w}) &= \hat{y}(\mathbf{x}_n, \mathbf{w}) \\ p(y_n = 0 \mid \mathbf{x}_n, \mathbf{w}) &= 1 - \hat{y}(\mathbf{x}_n, \mathbf{w}) \end{aligned}$$

- Rewritten:

$$p(y_n \mid \mathbf{x}_n, \mathbf{w}) = \hat{y}(\mathbf{x}_n, \mathbf{w})^{y_n} \{1 - \hat{y}(\mathbf{x}_n, \mathbf{w})\}^{1-y_n}$$

Loss function to be minimized

- Consider binary classification task using a single output unit with logistic sigmoid activation function:

$$\hat{y}(\mathbf{x}, \mathbf{w}) = h_{logistic}(a) = \frac{1}{1 + \exp(-a)}$$

- This defines a (Bernoulli) probability distribution over the label of each data point $\mathbf{x}_n$:

$$\begin{aligned} p(y_n = 1 \mid \mathbf{x}_n, \mathbf{w}) &= \hat{y}(\mathbf{x}_n, \mathbf{w}) \\ p(y_n = 0 \mid \mathbf{x}_n, \mathbf{w}) &= 1 - \hat{y}(\mathbf{x}_n, \mathbf{w}) \end{aligned}$$

- Rewritten:

$$p(y_n \mid \mathbf{x}_n, \mathbf{w}) = \hat{y}(\mathbf{x}_n, \mathbf{w})^{y_n} \{1 - \hat{y}(\mathbf{x}_n, \mathbf{w})\}^{1-y_n}$$

- Min. negative log likelihood of this distribution (aka cross entropy):

$$L(\mathbf{w}) = -\sum_{n=1}^{N} \{y_n \ln \hat{y}_n + (1 - y_n) \ln(1 - \hat{y}_n)\}$$

Loss function to be minimized

- For multiclass classification, use generalization of cross-entropy error:

$$L(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} y_{kn} \ln \hat{y}_k(\mathbf{x}_n, \mathbf{w})$$

- For regression, e.g., use squared error function:

$$L(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{\hat{y}(\mathbf{x}_n, \mathbf{w}) - y_n\}^2$$

Optimizing a loss / error function

- Given training data $\mathcal{D} = \langle(\mathbf{x}_i, y_i)\rangle_{i=1}^{N}$ and topology of an MLP
- Task: adapt weights w to minimize the loss:

$$\underset{\mathbf{w}}{minimize}\ L(\mathbf{w}; \mathcal{D})$$

- Interpret L just as a mathematical function depending on $\mathbf{w}$ and forget about its semantics, then we are faced with a problem of mathematical optimization

Lecture Overview

Optimization theory

- Discusses mathematical problems of the form:

$$\underset{\mathbf{u}}{minimize}\ f(\mathbf{u}),$$

where $\vec{u}$ is any vector of suitable size.

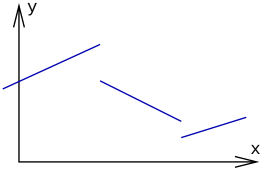
Optimization theory

- Discusses mathematical problems of the form:

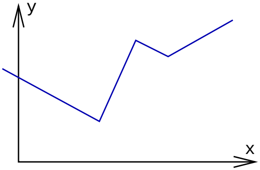
$$\underset{\mathbf{u}}{minimize}\ f(\mathbf{u}),$$

where $\vec{u}$ is any vector of suitable size.

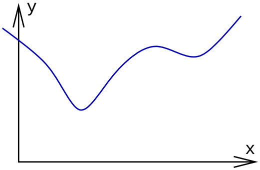
- Simplification: here, we only consider functions f which are continuous and differentiable



non continuous function
(disrupted)



continuous, non differentiable function (folded)



differentiable function
(smooth)

Optimization theory (cont.)

- A global minimum $\vec{u}^*$ is a point such that:

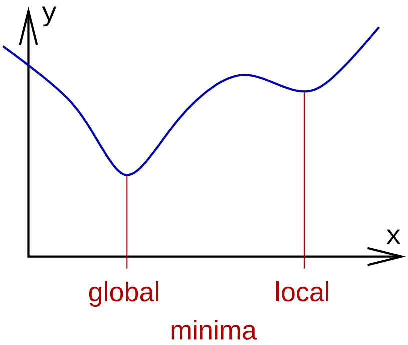
$$f(\mathbf{u}^*) \leq f(\mathbf{u})$$

  for all $\mathbf{u}$.

- A local minimum $\mathbf{u}^+$ is a point such that exist $r > 0$ with

$$f(\mathbf{u}^+) \leq f(\mathbf{u})$$

  for all points $\vec{u}$ with $||\vec{u} - \vec{u}^+|| < r$



Optimization theory (cont.)

- Analytical way to find a minimum:
  For a local minimum $\mathbf{u}^+$, the gradient of f becomes zero:

$$\frac{\partial f}{\partial u_i}(\mathbf{u}^+) = 0 \quad \text{for all } i$$

  Hence, calculating all partial derivatives and looking for zeros is a good idea

Optimization theory (cont.)

- Analytical way to find a minimum:
  For a local minimum $\mathbf{u}^+$, the gradient of f becomes zero:

  $$\frac{\partial f}{\partial u_i}(\mathbf{u}^+) = 0 \quad \text{for all } i$$

  Hence, calculating all partial derivatives and looking for zeros is a good idea

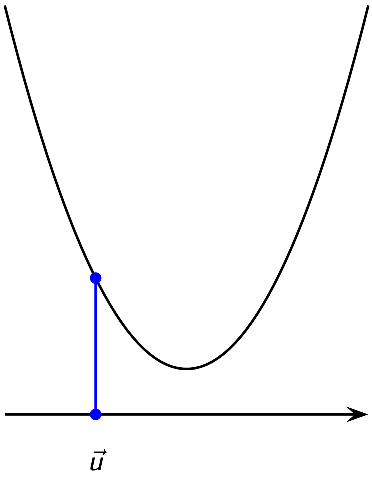
- But: for neural networks, we can't write down a solution for the minimization problem in closed form
  - even though $\frac{\partial f}{\partial u_i} = 0$ holds at (local) solution points

  $\rightarrow$ need to resort to iterative methods

Optimization theory (cont.)

- Numerical way to find a minimum, searching:
  assume we start at point $\mathbf{u}$.

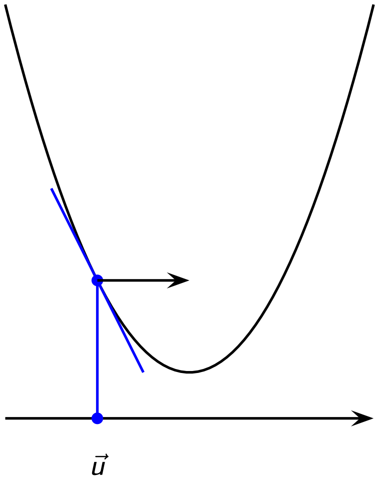
  Which is the best direction to search for a point $\mathbf{v}$ with $f(\mathbf{v}) < f(\mathbf{u})$?



Optimization theory (cont.)

- Numerical way to find a minimum, searching:
  assume we start at point $\mathbf{u}$.

  Which is the best direction to search for a point $\mathbf{v}$ with $f(\mathbf{v}) < f(\mathbf{u})$?

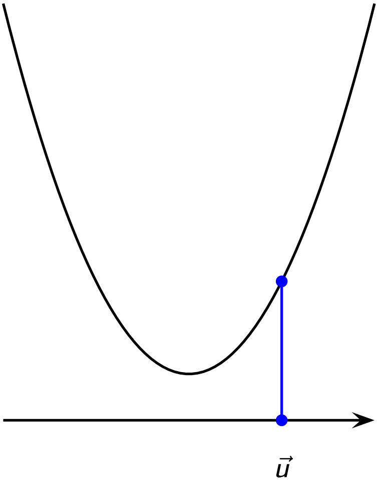


slope is negative (descending), go right!

Optimization theory (cont.)

- Numerical way to find a minimum, searching:
  assume we start at point $\mathbf{u}$.

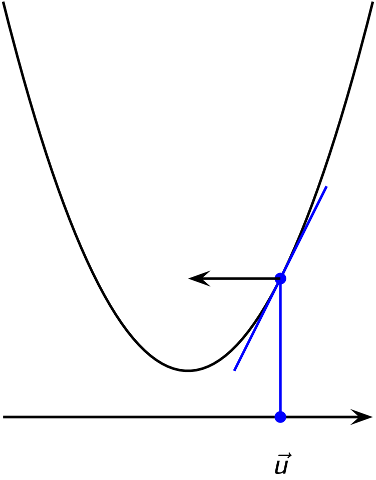
  Which is the best direction to search for a point $\mathbf{v}$ with $f(\mathbf{v}) < f(\mathbf{u})$?



Optimization theory (cont.)

- Numerical way to find a minimum, searching:
  assume we start at point $\mathbf{u}$.

  Which is the best direction to search for a point $\mathbf{v}$ with $f(\mathbf{v}) < f(\mathbf{u})$?



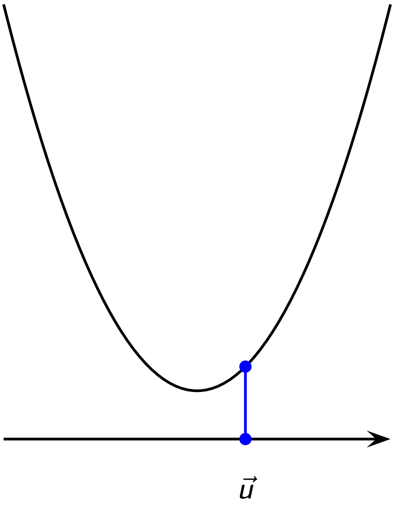
slope is positive (ascending), go left!

Optimization theory (cont.)

- Numerical way to find a minimum, searching:
  assume we start at point $\mathbf{u}$.

  Which is the best direction to search for a point $\mathbf{v}$ with $f(\mathbf{v}) < f(\mathbf{u})$?

  Which is the best stepwidth?

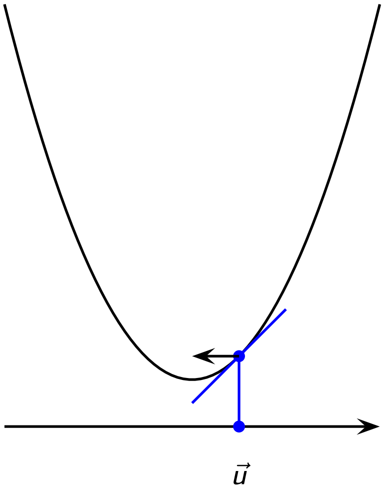


Optimization theory (cont.)

- Numerical way to find a minimum, searching:
  assume we start at point $\mathbf{u}$.

  Which is the best direction to search for a point $\mathbf{v}$ with $f(\mathbf{v}) < f(\mathbf{u})$?

  Which is the best stepwidth?



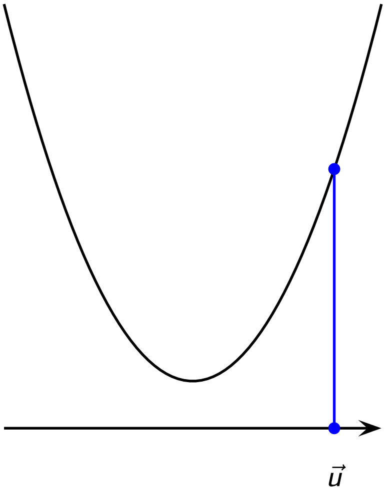
slope is small, small step!

Optimization theory (cont.)

- Numerical way to find a minimum, searching:
  assume we start at point $\mathbf{u}$.

  Which is the best direction to search for a point $\mathbf{v}$ with $f(\mathbf{v}) < f(\mathbf{u})$?

  Which is the best stepwidth?

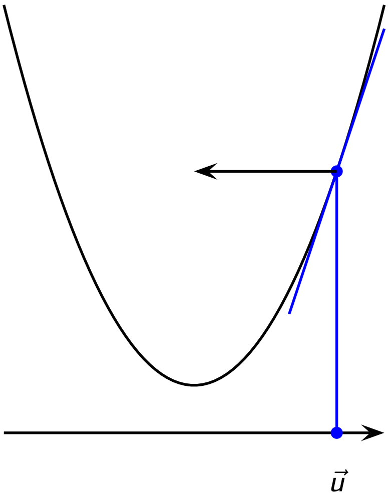


Optimization theory (cont.)

- Numerical way to find a minimum, searching:
  assume we start at point $\mathbf{u}$.

  Which is the best direction to search for a point $\mathbf{v}$ with $f(\mathbf{v}) < f(\mathbf{u})$?

  Which is the best stepwidth?



slope is large, large step!

Optimization theory (cont.)

- Numerical way to find a minimum, searching:
  assume we start at point $\mathbf{u}$.

  Which is the best direction to search for a point $\mathbf{v}$ with $f(\mathbf{v}) < f(\mathbf{u})$?

  Which is the best stepwidth?
- general principle:

$$v_i \leftarrow u_i - \epsilon \frac{\partial f}{\partial u_i}$$

$\epsilon > 0$ is called learning rate

Gradient descent

- Gradient descent approach:

Require: mathematical function f, learning rate $\epsilon > 0$
Ensure: returned vector is close to a local minimum of f

1: choose an initial point $\mathbf{u}$
2: **while** $||\nabla_{\mathbf{u}} f(\mathbf{u})||$ not close to 0 **do**
3: $\quad \mathbf{u} \leftarrow \mathbf{u} - \epsilon \cdot \nabla_{\mathbf{u}} f(\mathbf{u})$
4: **end while**
5: **return** $\mathbf{u}$

- Note: $\nabla_{\mathbf{u}} f := [\frac{\partial f}{\partial u_1}, \ldots, \frac{\partial f}{\partial u_K}]$ for K-dimensionsal $\mathbf{u}$

Calculating partial derivatives

- Our typical loss functions are defined across data points:

$$L(\mathbf{w}) = \sum_{n=1}^{N} L_n(\mathbf{w}) = L(f(\mathbf{x_n};\mathbf{w}), y_n)$$

Calculating partial derivatives

- Our typical loss functions are defined across data points:

$$L(\mathbf{w}) = \sum_{n=1}^{N} L_n(\mathbf{w}) = L(f(\mathbf{x_n}; \mathbf{w}), y_n)$$

- We can compute their partial derivatives as a sum over data points:

$$\frac{\partial L}{\partial w_j} = \sum_{n=1}^{N} \frac{\partial L_n}{\partial w_j}$$

Calculating partial derivatives

- Our typical loss functions are defined across data points:

$$L(\mathbf{w}) = \sum_{n=1}^{N} L_n(\mathbf{w}) = L(f(\mathbf{x_n}; \mathbf{w}), y_n)$$

- We can compute their partial derivatives as a sum over data points:

$$\frac{\partial L}{\partial w_j} = \sum_{n=1}^{N} \frac{\partial L_n}{\partial w_j}$$

- The method of backpropagation makes consistent use of the chain rule of calculus to compute the partial derivatives $\frac{\partial L_n}{\partial w_j}$ w.r.t. each network weight w_j, re-using previously computed results
  - Backpropagation is not covered here, but, e.g., in ML lecture

Do we need gradients based on the entire data set?

- Using the entire set is referred to as batch gradient descent

Do we need gradients based on the entire data set?

- Using the entire set is referred to as batch gradient descent
- Gradients get more accurate when based on more data points
  - But using more data has diminishing returns w.r.t reduction in error
  - Usually faster progress by updating more often based on cheaper, less accurate estimates of the gradient

Do we need gradients based on the entire data set?

- Using the entire set is referred to as batch gradient descent
- Gradients get more accurate when based on more data points
  - But using more data has diminishing returns w.r.t reduction in error
  - Usually faster progress by updating more often based on cheaper, less accurate estimates of the gradient
- Common approach in practice: compute gradients over mini-batches
  - Mini-batch: small subset of the training data
  - Today, this is commonly called stochastic gradient descent (SGD)

Stochastic gradient descent

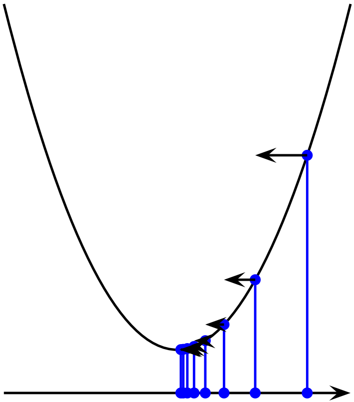
- Stochastic gradient descent (SGD)

Require: mathematical function f, learning rate $\epsilon > 0$
Ensure: returned vector is close to a local minimum of f

choose an initial point $\mathbf{w}$
while stopping criterion not met **do**
    Sample a minibatch of m examples $\mathbf{x^{(1)}}, \ldots, \mathbf{x^{(m)}}$ with corresponding targets $\mathbf{y}^{(i)}$ from the training set
    Compute gradient $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\mathbf{w}} \sum_{i=1}^{m} L(f(\mathbf{x^{(i)}}; \mathbf{w}), \mathbf{y^{(i)}})$
    Update parameter: $\mathbf{w} \leftarrow \mathbf{w} - \epsilon \cdot \mathbf{g}$
end while
return $\mathbf{w}$

Problems with suboptimal choices for learning rate

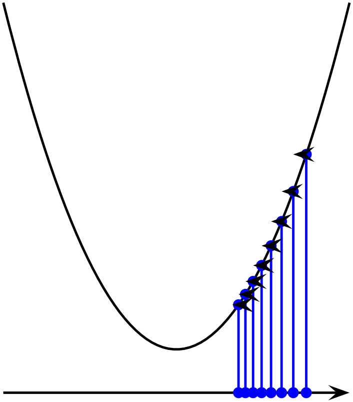
- choice of ϵ
  1. case small ϵ: convergence



Problems with suboptimal choices for learning rate

- choice of ϵ

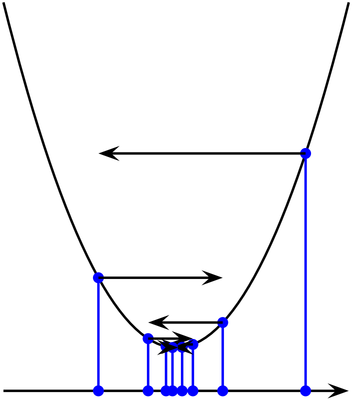
  2. case very small ϵ: convergence, but it may take very long



Problems with suboptimal choices for learning rate

- choice of ϵ

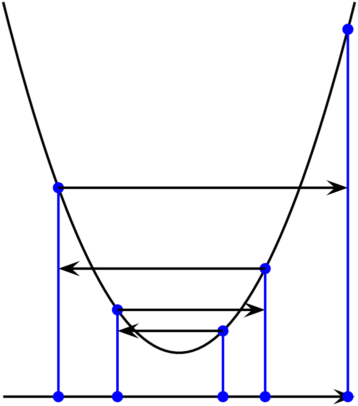
  3. case medium size ϵ: convergence



Problems with suboptimal choices for learning rate

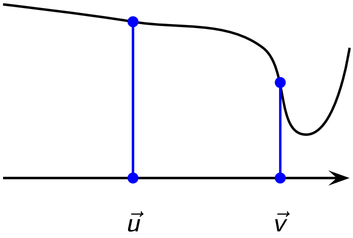
- choice of ϵ

  4. case large ϵ: divergence



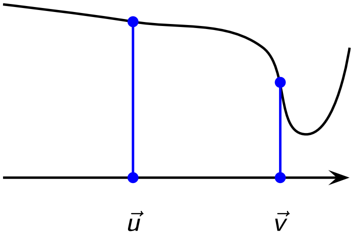
Other reasons for problems with gradient descent

- flat spots and steep valleys:
  need larger ϵ in $\vec{u}$ to jump over the uninteresting flat area but need smaller ϵ in $\vec{v}$ to meet the minimum

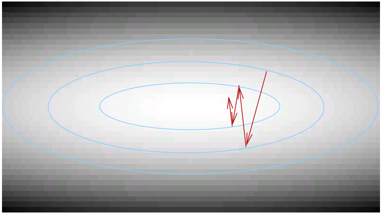


Other reasons for problems with gradient descent

- flat spots and steep valleys:
  need larger ϵ in $\vec{u}$ to jump over the uninteresting flat area but need smaller ϵ in $\vec{v}$ to meet the minimum

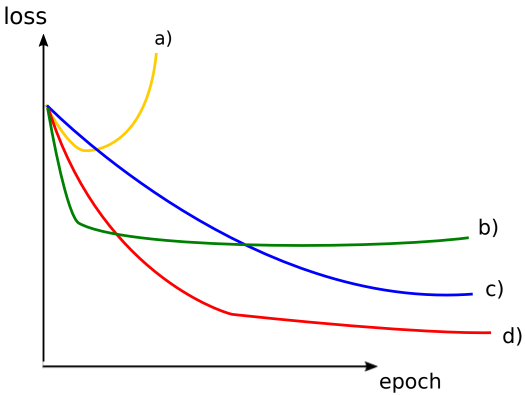


- zig-zagging:
  in higher dimensions: ϵ is not appropriate for all dimensions



Learning rate quizz

Which curve denotes low, high, very high, and good learning rate?



Gradient descent – Conclusion

- Pure gradient descent is a nice framework
- In practice, stochastic gradient descent is used
- Finding the right learning rate ϵ is tedious

Gradient descent – Conclusion

- Pure gradient descent is a nice framework
- In practice, stochastic gradient descent is used
- Finding the right learning rate ϵ is tedious

Heuristics to overcome problems of gradient descent:

- Gradient descent with momentum
- Individual learning rates for each dimension
- Adaptive learning rates
- Decoupling steplength from partial derivates

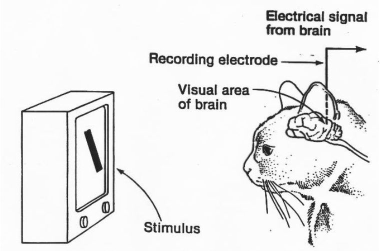
Lecture Overview

Lecture Overview

Historical context and inspiration from Neuroscience

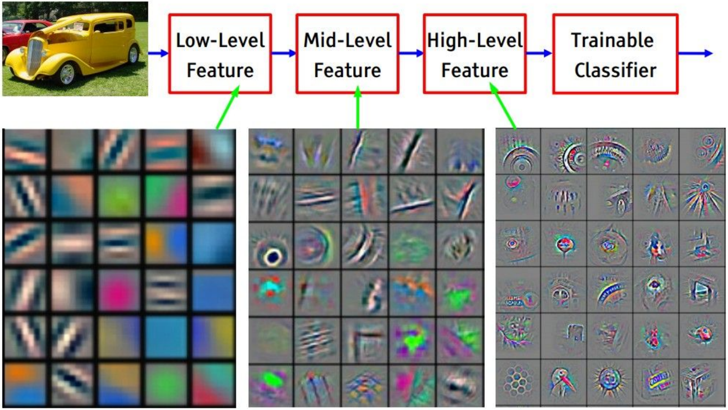
Hubel & Wiesel (Nobel prize 1981) found in several studies in the 1950s and 1960s:

- Visual cortex has feature detectors (e.g., cells with preference for edges with specific orientation)
  - edge location did not matter
- Simple cells as local feature detectors
- Complex cells pool responses of simple cells
- There is a feature hierarchy



Learned feature hierarchy

[From recent Yann LeCun slides]



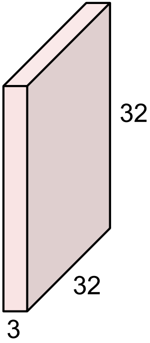
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

[slide credit: Andrej Karpathy]

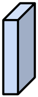
Convolutions illustrated

Filters always extend the full depth of the input volume

32x32x3 image



5x5x3 filter



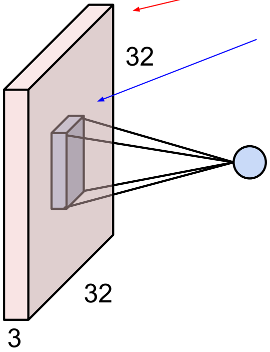
Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

[slide credit: Andrej Karpathy]

Convolutions illustrated (cont.)

32x32x3 image

5x5x3 filter w

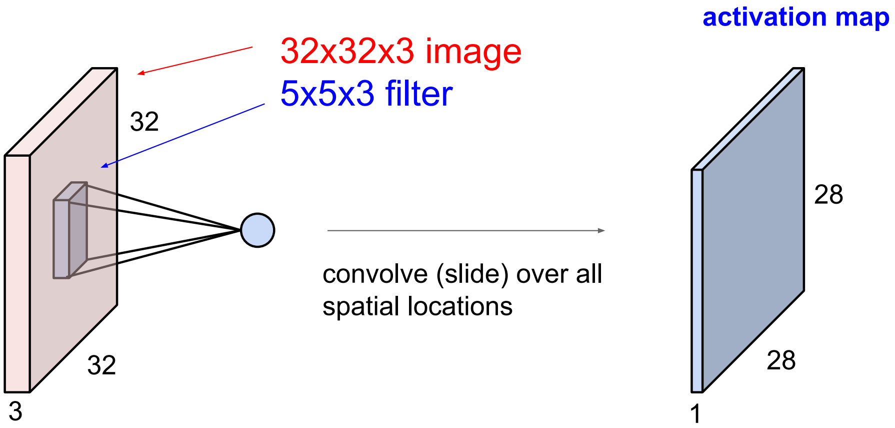


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

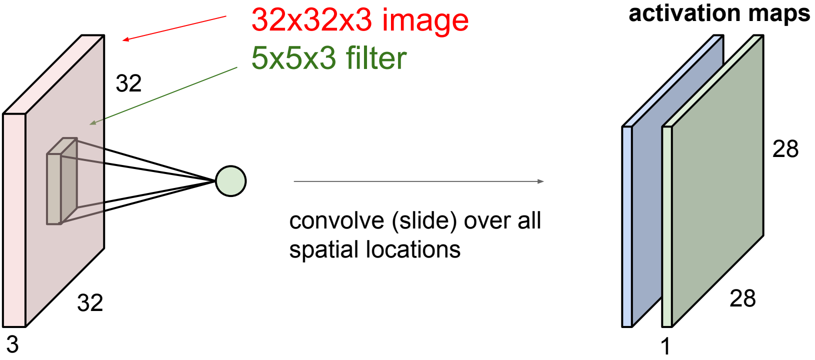
[slide credit: Andrej Karpathy]

Convolutions illustrated (cont.)



[slide credit: Andrej Karpathy]

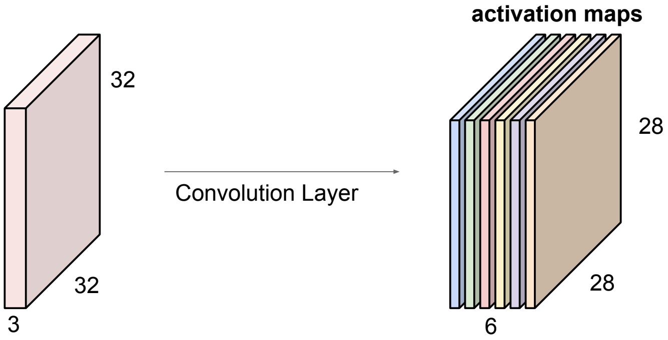
consider a second, green filter



[slide credit: Andrej Karpathy]

Convolutions – several filters

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

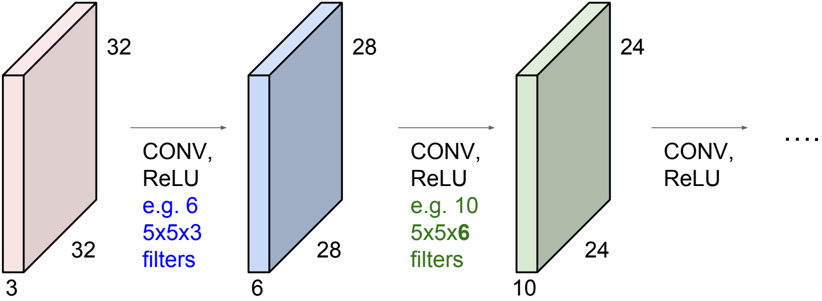


We stack these up to get a "new image" of size 28x28x6!

[slide credit: Andrej Karpathy]

Stacking several convolutional layers

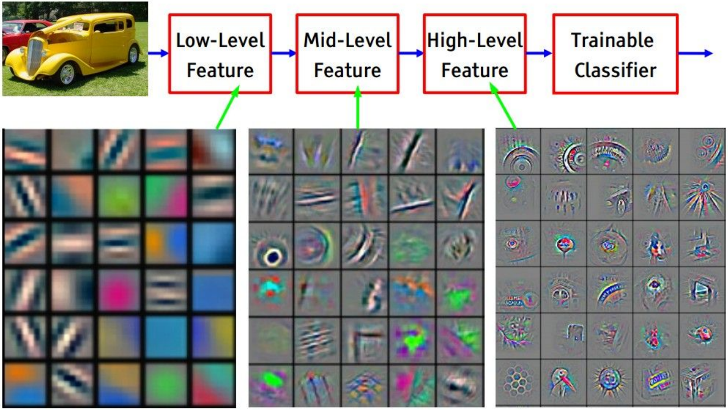
Convolutional layers stacked in a ConvNet



[slide credit: Andrej Karpathy]

Learned feature hierarchy

[From recent Yann LeCun slides]

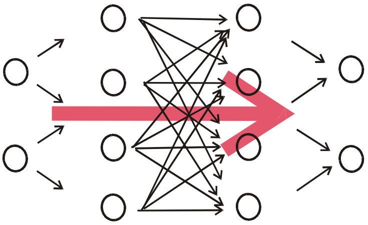


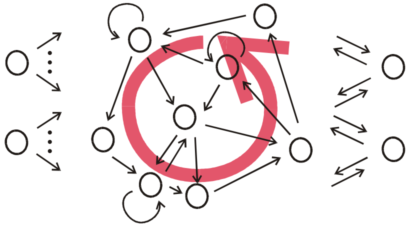
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

[slide credit: Andrej Karpathy]

Lecture Overview

Feedforward vs Recurrent Neural Networks





[Source: Jaeger, 2001]

Recurrent Neural Networks (RNNs)

- Neural Networks that allow for cycles in the connectivity graph

Recurrent Neural Networks (RNNs)

- Neural Networks that allow for cycles in the connectivity graph
- Cycles let information persist in the network for some time (state), and provide a time-context or (fading) memory

Recurrent Neural Networks (RNNs)

- Neural Networks that allow for cycles in the connectivity graph
- Cycles let information persist in the network for some time (state), and provide a time-context or (fading) memory
- Very powerful for processing sequences

Recurrent Neural Networks (RNNs)

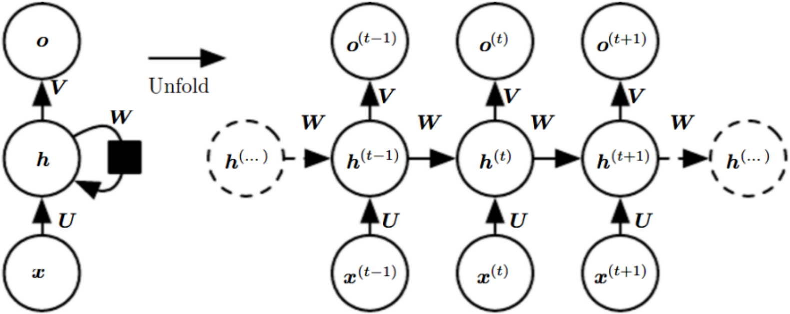
- Neural Networks that allow for cycles in the connectivity graph
- Cycles let information persist in the network for some time (state), and provide a time-context or (fading) memory
- Very powerful for processing sequences
- Implement dynamical systems rather than function mappings, and can approximate any dynamical system with arbitrary precision

Recurrent Neural Networks (RNNs)

- Neural Networks that allow for cycles in the connectivity graph
- Cycles let information persist in the network for some time (state), and provide a time-context or (fading) memory
- Very powerful for processing sequences
- Implement dynamical systems rather than function mappings, and can approximate any dynamical system with arbitrary precision
- They are Turing-complete [Siegelmann and Sontag, 1991]

Abstract schematic

With fully connected hidden layer:



[Goodfellow et al'2016]

one to many

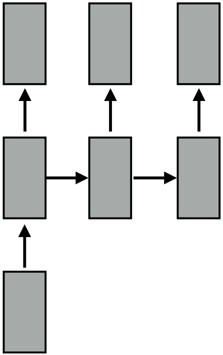
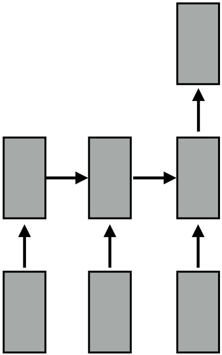


image caption generation

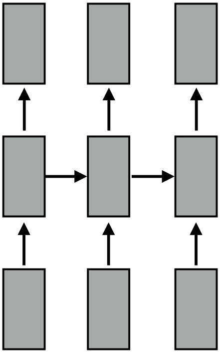
many to one



temporal classification

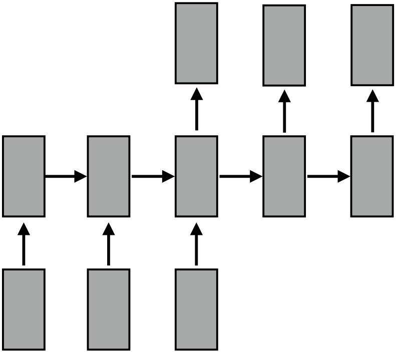
Sequence to sequence mapping (cont.)

many to many



video
frame labeling

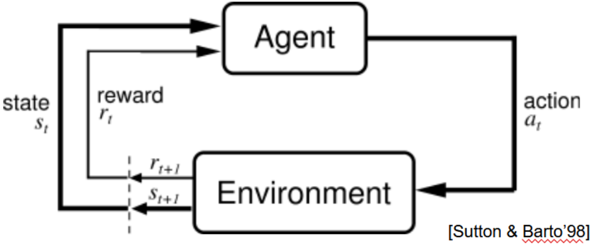
many to many



automatic
translation

Lecture Overview

Reinforcement Learning



- Finding optimal policies for MDPs
- Reminder: states $s \in S$, actions $a \in A$, transition model T, rewards r
- Policy: complete mapping $\pi : S \rightarrow A$ that specifies for each state s which action $\pi(s)$ to take

Deep Reinforcement Learning

- Policy-based deep RL
  - Represent policy $\pi : S \rightarrow A$ as a deep neural network with weights w
  - Evaluate w by "rolling out" the policy defined by w
  - Optimize weights to obtain higher rewards (using approx. gradients)
  - Examples: AlphaGo & modern Atari agents

Deep Reinforcement Learning

- Policy-based deep RL
  - Represent policy $\pi : S \rightarrow A$ as a deep neural network with weights w
  - Evaluate w by "rolling out" the policy defined by w
  - Optimize weights to obtain higher rewards (using approx. gradients)
  - Examples: AlphaGo & modern Atari agents

- Value-based deep RL
  - Basically value iteration, but using a deep neural network (= function approximator) to generalize across many states and actions
  - Approximate optimal state-value function $U(s)$
    or state-action value function $Q(s, a)$

Deep Reinforcement Learning

- Policy-based deep RL
  - Represent policy $\pi : S \to A$ as a deep neural network with weights w
  - Evaluate w by "rolling out" the policy defined by w
  - Optimize weights to obtain higher rewards (using approx. gradients)
  - Examples: AlphaGo & modern Atari agents
- Value-based deep RL
  - Basically value iteration, but using a deep neural network (= function approximator) to generalize across many states and actions
  - Approximate optimal state-value function $U(s)$ or state-action value function $Q(s, a)$
- Model-based deep RL
  - If transition model T is not known
  - Approximate T with a deep neural network (learned from data)
  - Plan using this approximate transition model

Deep Reinforcement Learning

- Policy-based deep RL
  - Represent policy $\pi : S \rightarrow A$ as a deep neural network with weights w
  - Evaluate w by "rolling out" the policy defined by w
  - Optimize weights to obtain higher rewards (using approx. gradients)
  - Examples: AlphaGo & modern Atari agents

- Value-based deep RL
  - Basically value iteration, but using a deep neural network (= function approximator) to generalize across many states and actions
  - Approximate optimal state-value function $U(s)$
    or state-action value function $Q(s, a)$

- Model-based deep RL
  - If transition model T is not known
  - Approximate T with a deep neural network (learned from data)
  - Plan using this approximate transition model

→ Use deep neural networks to represent policy / value function / model

Lecture Overview

An Exciting Approach to AI: Learning as an Alternative to Traditional Programming

- We don't understand how the human brain solves certain problems
  - Face recognition
  - Speech recognition
  - Playing Atari games
  - Picking the next move in the game of Go
- We can nevertheless learn these tasks from data/experience

An Exciting Approach to AI: Learning as an Alternative to Traditional Programming

- We don't understand how the human brain solves certain problems
  - Face recognition
  - Speech recognition
  - Playing Atari games
  - Picking the next move in the game of Go
- We can nevertheless learn these tasks from data/experience
- If the task changes, we simply re-train

An Exciting Approach to AI: Learning as an Alternative to Traditional Programming

- We don't understand how the human brain solves certain problems
  - Face recognition
  - Speech recognition
  - Playing Atari games
  - Picking the next move in the game of Go
- We can nevertheless learn these tasks from data/experience
- If the task changes, we simply re-train
- We can construct computer systems that are too complex for us to understand anymore ourselves...
  - E.g., deep neural networks have millions of weights.
  - E.g., AlphaGo, the system that beat world champion Lee Sedol
    - + David Silver, lead author of AlphaGo cannot say why a move is good
    - + Paraphrased: "You would have to ask a Go expert."

Summary: Why is Deep Learning so Popular?

- Excellent empirical results in many domains
  - very scalable to big data
  - but beware: not a silver bullet

Summary: Why is Deep Learning so Popular?

- Excellent empirical results in many domains
  - very scalable to big data
  - but beware: not a silver bullet
- Analogy to the ways humans process information
  - mostly tangential

Summary: Why is Deep Learning so Popular?

- Excellent empirical results in many domains
  - very scalable to big data
  - but beware: not a silver bullet
- Analogy to the ways humans process information
  - mostly tangential
- Allows end-to-end learning
  - no more need for many complicated subsystems
  - e.g., dramatically simplified Google's translation

Summary: Why is Deep Learning so Popular?

- Excellent empirical results in many domains
  - very scalable to big data
  - but beware: not a silver bullet
- Analogy to the ways humans process information
  - mostly tangential
- Allows end-to-end learning
  - no more need for many complicated subsystems
  - e.g., dramatically simplified Google's translation
- Very versatile/flexible
  - easy to combine building blocks
  - allows supervised, unsupervised, and reinforcement learning

Lots of Work on Deep Learning in Freiburg

- Computer Vision (Thomas Brox)
  - Images, video
- Robotics (Wolfram Burgard)
  - Navigation, grasping, object recognition
- Neurorobotics (Joschka Boedecker)
  - Robotic control
- Machine Learning (Frank Hutter)
  - Optimization of deep nets, learning to learn
- Neuroscience (Tonio Ball, Michael Tangermann, and others)
  - EEG data and other applications from BrainLinks-BrainTools

→ Details when the individual groups present their research

Summary by learning goals

Having heard this lecture, you can now . . .

- Explain the terms representation learning and deep learning
- Describe the main principles behind MLPs
- Describe how neural networks are optimized in practice
- On a high level, describe
  - Convolutional Neural Networks
  - Recurrent Neural Networks
  - Deep Reinforcement Learning