Module 4: Machine Learning

Girish Chowdhary and Karan Chawla

Assistant Professor Director Distributed Autonomous Systems Lab
Coordinated Science Lab (CSL), Agricultural and Biological Enginering, Aerospace Engineering, Electrical and Compute Engineering, IGB, Beckman
University of Illinois at Urbana Champaign,
http://daslab.illinois.edu

Thanks to Alex Schwing for sharing some of his Deep Learning Slides

February 15, 2018

Overview

Part I – Machine Learning

- Regression: Basics
- Kernel based regression: Radial Basis Function Networks and Gaussian Processes
- Single Hidden Layer (2-layer) Neural Networks

Part II – Deep Learning

- Classification, and the need to go beyond Gaussian kernel models
- Convolutional Neural Networks
- Implementation

Part III – Practical Deep Learning

- Keras/Pytorch Introduction
- Classification on a real dataset
- Unsupervised Learning
- RNNs and GANs

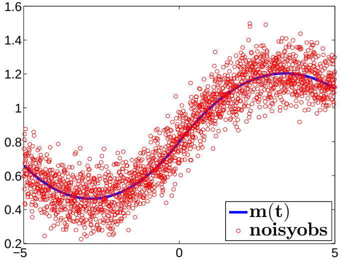
References

- Murphy Kevin, Machine Learning a Probabilistic Perspective [6]: A great primer on ML
- Goodfellow et al. Deep Learning [4]: Some details of state of the art
- The structure of this module: Lectures and slides in class by Girish; demos, experiments, and hacks by Karan

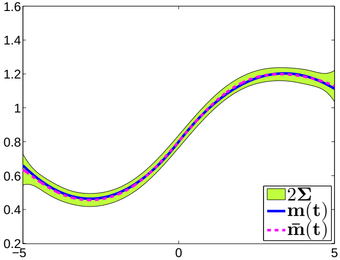
Notation

- y: Dependent variable, output variable, the variable to learn. $y \in \mathbb{R}$ for regression, or in $y \in \{-1, 1\}$ for binary classification. y_i is the i^{th} training sample
- x: Independent variable, input variable. $x \in \mathbb{R}^n$ for regression, $x \in D$ for classification
- W, V: Weights
- θ: A generic container for weights, e.g. $\theta = [W, V]$
- $\mathbf{f}$ bases functions, not to be confused with f

- Finding patterns in data
- Building models from data $\rightarrow$ being able to predict patterns
- Deterministic vs Probabilistic models



(a) noisy data



(b) Probablistic model

Figure:

Simple example: Curve fitting

- Consider that we are given a noisy data set $S = \{x_1, x_2, ..., x_N\}$
- Goal: Fit a curve for the given data: simple case: polynomial fit

$$y(x, w) = w_0 + w_1 x + w_2 x^2 + ... + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

- can write this in vector form: $y(x, w) = [1 \; x \; x^2 \; x^3 \; ... \; x^M]W$, where $W \in \Re^M$ is a column vector of weights
- Alternatively let $\phi(x) = [1 \; x \; x^2 \; x^3 \; ... \; x^M]^T$, then $y(x, W) = W^T \phi(x)$
- We can define a least squares error function:

$$E(W) = \frac{1}{2} \sum_{n=1}^{N} \{(y(x_n, W) - W^T \phi(x_n))^2\}$$

- Agent can find weights W to minimize the cost function

Simple example

- Issues: what should be the dimension of W? $\rightarrow$ the complexity of our model
- Will our approach handle noise?
- What other basis functions ϕ can I use?

- Heuristic: If the value of W is too high we will get overfitting

- solution: we can avoid overfitting through regularization: penalize high values in W:

$$\tilde{E}(w) = \frac{1}{2} \sum_{n=1}^{N} \{(y(x_n, W) - W^T \phi(x_n))^2\} + \frac{1}{2} \|W\|^2$$

- This still does not solve the problem of how to choose the number of parameters M in our model (too little, too much?)

- Bayesian (nonparametric) approach: adapt the number of parameters to the data

- Deep Neural Network approach: Use overparameterized models

True function

Let $y \in \mathbb{R}$ be the dependent variable (scalar for now, but the idea generalizes immediately), and $x \in \mathbb{R}^n$ be the independent variable

$$y = f(x) \tag{1}$$

Is the true unknown function that satisfies some problem-specific guarantees

For $x \in \mathbb{R}^n$, let m be the number of "features", i.e. $\phi(x) \in \mathbb{R}^m$, and $W^* \in \mathbb{R}^m$ a vector of "ideal" weights

Universal Function approximation theorems NN [5], RBFN [8]

$$\|y(x) - {W^*}^T \phi(x)\|_\infty < \epsilon(x) \tag{2}$$

The Machine learning problem

Let $L(y(x), W^T\phi(x))$ be a loss function, let D be the set of N training inputs x_n then solve

$$W = \arg\min_W L(y(x_n), W^T\phi(x_n)) \forall x_n \in D \tag{3}$$

Let $x \in \mathbb{R}^n$ be the independent variable, let $c_j \in \mathbb{R}^n$ be an "RBF center", then

Gaussian Radial Basis Functions

$$\phi(x, c_j) = e^{\frac{-\|x - c_j\|^2}{\sigma^2}} \tag{4}$$

Here σ is the bandwidth of the kernel (how fat the kernel is)

Let $\Phi(x) = [\phi(x, c_1), \phi(x, c_2), \cdots, \phi(x, c_m)]^T$The Radial Basis Function Network (RBFN) is formulated as

RBFN

$$\hat{y}(x) = \sum_{j=1}^{m} w_i \phi(x, c_j) = W^T \Phi(x) \tag{5}$$

Let $H = [y_1, y_2, \cdots, y_N]^T$, $X = [\Phi(x_1), \Phi(x_2), \cdots, \Phi(x_N)]^T$ and let $L(W, x) = \|y(x) - W^T \Phi(x)\|_2^2$

Optimal (least-squares) solution for RBFN

$$W = \left[XX^T\right]^{-1} XH \tag{6}$$

Regularization

How do we prevent the Neural Network weights becoming "too big"?

Regularized Regression

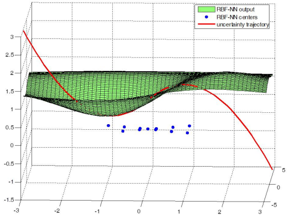
$$L(W, x) = \|y(x) - W^T \Phi(x)\|_2^2 + C\|W\|_2^2$$

Solution

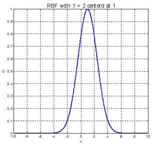
Optimal weights

$$W = \left[XX^T + CI\right]^{-1} XH$$

- RBFs: linear-in-the-parameters, easier to analyze and easier to tune, more popular
- How many bases functions to use?
- How do I know my model is doing "good"?
- Gaussian Processes: Generalizing the idea of RBFN to accommodate potentially infinite number of bases functions

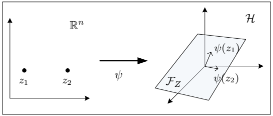


(a)



(b) A one-dimensional RBF kernel

- A **Mercer Kernel** $k(x_1, x_2) \in \mathbb{R}$ is a continuous, symmetric, positive definite function for $x_1, x_2 \in D \subset \mathbb{R}^n$
- Can think of kernels as a measure on similarity of any two points
- The Gaussian RBF is a Mercer kernel: $k(., x_2)k(., c_i) = \phi(., c_i) = e^{\frac{-\|x-c_j\|^2}{\sigma^2}}$



;

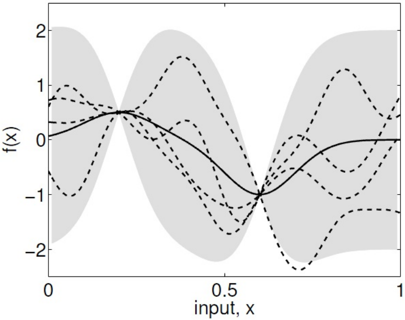
Figure: An example Hilbert space mapping.

(Mercer) RKHS

There exists a high (infinite) dimensional Hilbert space H and a mapping $\psi : D \to H$

- Kernel trick: Although ψ may be unknown: $k(x_1, x_2) = \langle \psi(x_1), \psi(x_2) \rangle$

- Idea: *Starting from an infinite set of possible uncertainty representations, use data to narrow down the most applicable subset* $\rightarrow$ Infer both the model structure and parameters from data
- Naive GP: add a kernel at every data point observed, i.e. add every $x(t)$ to the set of centers $C = [c_1, c_2, \cdots, c_m]^T$
- Issue: Large computational burden ($O(N^3)$) as data size increases, infeasible online
- Online sparsification: Only keep the most relevant kernels
  - Csato's online sparse GP [?]
  - Keep a buffer of most recent points



(b), posterior

Figure: Posterior estimate given GP assumption

- Let $X_t = \{x_1, \ldots, x_t\}$ be a set of state measurements, with outputs $y_t(x_i) = f(x_i) + \epsilon_i$, where $\epsilon_i \sim \mathcal{N}(0, \omega^2)$ is Gaussian noise.
- Under GP model, data $\{f(x_1), \ldots, f(x_t)\}$ has prior distribution $\mathcal{N}(0, K(X_t, X_t))$, where $K(X_t, X_t)$ is Gram matrix of the elements in X_t.
- Can be shown that given a new input x_{t+1}, joint distribution of the data:

$$\begin{bmatrix} y_t \\ y_{t+1} \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(X_t, X_t) + \omega^2 I & k_{x_{t+1}} \\ k_{x_{t+1}}^T & k_{t+1}^* \end{bmatrix}\right), \tag{7}$$

where $k_{x_{t+1}} = K(x_{t+1}, X_t)$ and $k_{t+1}^* = k(x_{t+1}, x_{t+1})$. Posterior:

$$p(y_{t+1} | X_t, y_t, x_{t+1}) \sim \mathcal{N}(\hat{m}_{t+1}, \bar{\Sigma}_{t+1}), \tag{8}$$

where

$$\hat{m}_{t+1} = w_{t+1}^T k_{x_{t+1}} \tag{9}$$

$$\bar{\Sigma}_{t+1} = k_{t+1}^* - k_{x_{t+1}}^T C k_{x_{t+1}} \tag{10}$$

are estimated mean and covariance respectively, and $C := (K(X_t, X_t) + \omega^2 I)^{-1}$ and $w_{t+1} := Cy_t$. Inversions can be computed recursively to improve computational efficiency

GPs as time-varying RBFNs

The key idea is that the set of centers can change with time
Naive: Add a center for every data point
Sparse: Use a fixed dictionary of centers and swap the centers out with the most "useful" ones [3, 2]

Challenge

Well, we still haven't really solved the problem of how many kernels do we need, we just replaced it with creating a very large and flat model $\rightarrow$ Broad networks

Stationary vs Non-stationary Kernel Functions

Maybe we can solve the problem by using more complicated kernels?

- **Stationary** Squared Exponential (SE) Kernel: For $x_i, x_j \in \mathbb{R}^2$

$$k(x_i, x_j) = \exp((x_i - x_j)^T \Sigma^{-1} (x_i - x_j))$$

  where $\Sigma = \sigma^2 \cdot \mathbf{I}_2$ is the covariance matrix,

- **Non-stationary** SE Kernel [7]:

$$k(x_i, x_j) = |\Sigma_i|^{\frac{1}{4}} |\Sigma_j|^{\frac{1}{4}} |(\Sigma_i + \Sigma_j)/2|^{-\frac{1}{2}} \exp(-Q_{ij})$$
$$Q_{ij} = (x_i - x_j)^T ((\Sigma_i + \Sigma_j)/2)^{-1} (x_i - x_j),$$

  Σ_i is the covariance matrix at x_i, in isotropic case

$$\Sigma_i = \begin{bmatrix} \sigma^2_{(x_i)_1} & 0 \\ 0 & \sigma^2_{(x_i)_2} \end{bmatrix}$$

- **Solving for** Σ_i: Computationally expensive MCMC sampling.
- Infeasible for massive scale data

- The general idea can be expanded to different distributions:
- Bayesian Nonparametrics: Model structure (number of parameters) and parameter values concurrently inferred from data [10, 9, 1]
- Attracting significant interest in the last 8-10 years

Model	**Typical Application**
GP	Learning continuous functions
DP	Mixture models
BP	Shared latent feature models
HDP-HMM	Hidden Markov models
HDP-SLDS-HMM HDP-AR-HMM	Hybrid system models
BP-AR-HMM	Hybrid system models with shared features
HBP	Mixtures of latent feature models

- Challenges: Computational scalability, on-line (sequential) implementation
- The Bayesian perspective is elegant, but practicality is limited due to MCMC (not the focus of this workshop)

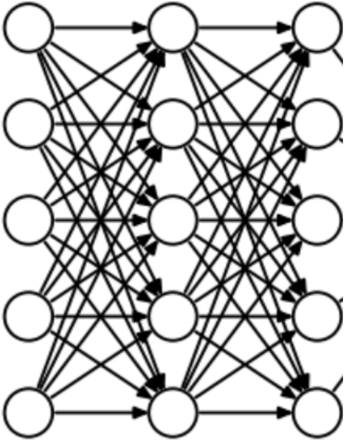
Adaptive Basis Functions

- Coming up with kernels is hard for general problems
- For regression, the Gaussian RBF does really well, in M3 we will show that it has some real power when dealing with smooth dynamical systems defined over continuous state-spaces
- But how do you build a kernel that compares two documents?
- Kernels that compare two images?
- There is another way: Adapt the features directly from the data
- That is learn filters that prioritize certain aspects of the data through some sort of a squashing function or gate

There is another way of handling this problem

What if we went deep instead of broad?

$$y = W^T \sigma(V^T \bar{x})$$



A more detailed look

$W, V, \bar{x}$ are defined in the following:

- n_3 denote the number of output layer neurons ($n_3 = 1$ for scalar case)
- n_2 denote the number of hidden layer neurons
- n_1 denote the number of input layer neurons
- $W \in \Re^{(n_2+1)\times n_3}$ is the NN synaptic weight matrix connecting the hidden layer with the output layer
- $V \in \Re^{(n_1+1)\times n_2}$ is the NN synaptic weight matrix connecting the input layer with the hidden layer

Universal Approximation Theorem [5]

Given an $\bar{\epsilon} > 0$, for all $\bar{x} \in D$, where D is a compact set, there exists a number of hidden layer neurons n_2, and an ideal set of weights (W^*, V^*) that brings the NN output to within an ϵ neighborhood of the function approximation error. The largest such ϵ is given by

$$\bar{\epsilon} = \sup_{\bar{x}\in D} \left\| {W^*}^T \sigma({V^*}^T \bar{x}) - f(\bar{x}) \right\|. \tag{11}$$

The Output layer weight matrix W

$$W = \begin{pmatrix} \Theta_{w,1} & \cdots & \Theta_{w,n_3} \\ w_{1,1} & \cdots & w_{1,n_3} \\ \vdots & \ddots & \vdots \\ w_{n_2,1} & \cdots & w_{n_2,n_3} \end{pmatrix} \in \Re^{(n_2+1)\times n_3}, \quad (12)$$

The hidden layer weight matrix V

$$V = \begin{pmatrix} \Theta_{v,1} & \cdots & \Theta_{v,n_2} \\ v_{1,1} & \cdots & v_{1,n_2} \\ \vdots & \ddots & \vdots \\ v_{n_1,1} & \cdots & w_{n_1,n_2} \end{pmatrix} \in \Re^{(n_1+1)\times n_2}, \quad (13)$$

The input vector with a bias term $\bar{x}$

$$\bar{x} = \begin{pmatrix} b_v \\ x_{in} \end{pmatrix} = \begin{pmatrix} b_v \\ x_{in_1} \\ x_{in_2} \\ \vdots \\ x_{in_{n_1}} \end{pmatrix} \in \Re^{n_1+1}. \quad (14)$$

let $a = V^T \bar{x} \in \Re_2^n$, and b_w denote the constant bias term usually set to 1 for the hidden layer neuron. Then the vector function $\sigma(a) \in \Re^{n_2+1}$ is given by

$$\sigma(a) = \begin{pmatrix} b_w \\ \sigma_1(a_1) \\ \vdots \\ \sigma_{n_2}(a_{n_2}) \end{pmatrix} \in \Re^{n_2+1}. \tag{15}$$

The elements of σ consist of sigmoidal activation functions, in which c_js are (different) constants

$$\sigma_j(a_j) = \frac{1}{1 + e^{-c_j a_j}} \tag{16}$$

$$\tag{17}$$

Alternative activation functions:

- tanh: $\sigma_j(a_j) = \tanh(a_j)$,
- RELU (Rectified Linear Activation Unit): $\sigma_j(a_j) = \max(0, a_j)$

Backpropagation

Genralize, let $\mathbf{f}$ denote σ (activation function)

Simplifying notation

$$\bar{x} \xrightarrow{V} a \xrightarrow{\mathbf{f}} z \xrightarrow{W} b \xrightarrow{h} y$$

- $a = V^T \bar{x}$
- $z = \mathbf{f}(a) = \mathbf{f}(V^T \bar{x})$
- $b = W^T z = W^T \mathbf{f}(V^T \bar{x})$
- Assume h is identity
- Let $\theta = W, V$

Generic loss

Let $L(\theta)$ denote the generic loss, recall for regression Negative Log Likelihood:

$$L(\theta) = -\sum_n \sum_k (\hat{y}_{nk}(\theta) - y_{nk})^2$$

For (k-class) classification Negative Log Likelihood (more late):

$$L(\theta) = -\sum_n \sum_k y_{nk} \log \hat{y}_{nk}(\theta)$$

- We need $\nabla_\theta L$

Let's start at the output, noting $\frac{\partial b}{\partial W} = \frac{\partial W^T z}{\partial W} = z$

$$\nabla_W L = \frac{\partial L}{\partial b} \nabla_W b = \frac{\partial L}{\partial b} z \tag{18}$$

Now note that

$$\frac{\partial L}{\partial b} \triangleq \delta^w = (\hat{y} - y) \tag{19}$$

So the overall gradient is the input before applying **f** times the error:

$$\nabla_W L = \delta^W z \tag{20}$$

Now noting that $\nabla_V a = \frac{\partial a}{\partial V} = \frac{\partial V^T \bar{x}}{\partial V} = \bar{x}$:

$$\nabla_V L = \frac{\partial L}{\partial a} \nabla_V a \triangleq \delta^V \bar{x} \tag{21}$$

What we need now is δ^V, for this we "backpropagate" this error:

$$\delta^v = \frac{\partial L}{\partial a} = \frac{\partial L}{\partial b}\frac{\partial b}{\partial a} = \delta^W \frac{\partial b}{\partial a} \tag{22}$$

Now $b = W^T z = W^T \mathbf{f}(a) = W^T \mathbf{f}(V^T \bar{x})$

$$\frac{\partial b}{\partial a} = \frac{\partial W^T z}{\partial a} = \frac{\partial W^T \mathbf{f}(a)}{\partial a}$$

Which brings us to

$$\frac{\partial b}{\partial a} = \frac{\partial W^T \mathbf{f}(a)}{\partial a} = W^T \mathbf{f}'(a) \tag{23}$$

And now we can use the definitions: The elements of σ consist of sigmoidal activation functions, which are given by

$$\mathbf{f} = \sigma_j(a_j) = \frac{1}{1 + e^{-c_j a_j}} \tag{24}$$

$$\mathbf{f}' = \frac{\partial}{\partial a_j}\sigma(a_j) = \sigma_j(a_j)(1 - \sigma_j(a_j)) \tag{25}$$

Alternative activation functions:

- tanh: $\mathbf{f}_j(a_j) = \tanh(a_j)$, $\frac{\partial}{\partial a_j}\mathbf{f}(a_j) = 1 - \tanh^2(a_j)$
- RELU (Rectified Linear Activation Unit): $\mathbf{f}_j(a_j) = \max(0, a_j)$, its derivative is defined everywhere except at 0 as $\mathbf{f}_j^{'}(a_j) = \begin{cases} 1, & \text{if } x > 0. \\ 0, & \text{otherwise.} \end{cases}$

Some NN activation functions

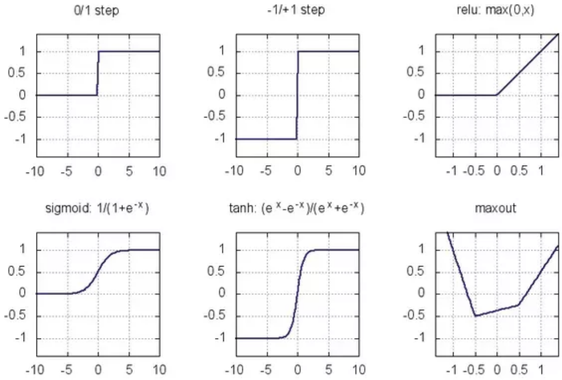


Figure: These ones are examples of squashers

(Figure: https://www.quora.com/Do-people-use-power-functions-like-x-3-or-x-5-as-activation-functions-in-artificial-neural-networks)

Question: How do we do classification ($y \in \{-1, 1\}$) $\Rightarrow$ Logistic Regression

Logistic Functions

Simple idea: Convert a linear function to a binary discriminator:

$$\mu(x) = \sigma(W^T x)$$

Where σ is a "squashing function". Examples:

- Sigmoidal
- tanh
- RELU

Probabilistic Intuition

$$p(y|x, W) = Ber(y|\mu(x))$$

where $\mu(x) = E[y|x] = p(y = 1|x)$

Consider the binary classification problem $y = \{-1, 1\}$Then

$$p(y=1) = \frac{1}{1+e^{-W^T x}}$$

$$p(y=-1) = \frac{1}{1+e^{W^T x}}$$

Hence

Negative Log Likelihood for Classification

$$L(W) = \sum_{i=1}^{N} \log(1 + e^{-y_i W^T x_i})$$

Cannot write down the Maximum Likelihood estimate in closed form $\Rightarrow$ Stochastic gradient descent

$$W_{k+1} = W_k + \Gamma d_k$$

Where $d_k = -H_k^{-1} g_k$, and H_k is the Hessian, g_k is the gradient of $L(w)$ (Murphy pp 249[6])

General Problem Formulation

Given a representer of the form $y = \mathbf{f}(\theta, x)$ and a generic loss function $L(\theta, x, y)$ solve

Generic ML problem

$$\min_{\theta} \sum_{x_i, y_i \in D} L(\mathbf{f}(\theta, x_i), y_i)$$

The generic loss function could also have robustifying terms, such as regularization: $||\theta||_2^2$

- How should we choose $\mathbf{f}$?

General Problem Formulation

Given a representer of the form $y = \mathbf{f}(\theta, x)$ and a generic loss function $L(\theta, x, y)$ solve

Generic ML problem

$$\min_{\theta} \sum_{x_i, y_i \in D} L(\mathbf{f}(\theta, x_i), y_i)$$

The generic loss function could also have robustifying terms, such as regularization: $||\theta||_2^2$

- How should we choose $\mathbf{f}$?
- We saw kernel models, they are great for regression, what about more general problems?

General Problem Formulation

Given a representer of the form $y = \mathbf{f}(\theta, x)$ and a generic loss function $L(\theta, x, y)$ solve

Generic ML problem

$$\min_{\theta} \sum_{x_i, y_i \in D} L(\mathbf{f}(\theta, x_i), y_i)$$

The generic loss function could also have robustifying terms, such as regularization: $||\theta||_2^2$

- How should we choose $\mathbf{f}$?
- We saw kernel models, they are great for regression, what about more general problems?
- We saw generic logistic regression problem, and its extension to the Single Hidden Layer network
- We noted we can "learn" θ using gradient descent, and backpropagate errors
- The idea we now explore is that of "adaptive data-driven bases", the essence of Deep Learning

PART II – Deep Learning

Deep Learning:

What function $\mathbf{f}(w, x, y)$ to choose?

Deep Learning:

What function $\mathbf{f}(w, x, y)$ to choose?

- Choose any differentiable composite function

$$\mathbf{f}(w, x, y) = f_1(w_1, y, f_2(w_2, f_3(\ldots f_n(w_n, x)\ldots)))$$

Deep Learning:

What function $\mathbf{f}(w, x, y)$ to choose?

- Choose any differentiable composite function

$$\mathbf{f}(w, x, y) = f_1(w_1, y, f_2(w_2, f_3(\ldots f_n(w_n, x)\ldots)))$$

- More generally: functions can be represented by a directed acyclic graph (computation graph)

Deep Learning:

What function $\mathbf{f}(w, x, y)$ to choose?

- Choose any differentiable composite function

$$\mathbf{f}(w, x, y) = f_1(w_1, y, f_2(w_2, f_3(\ldots f_n(w_n, x) \ldots)))$$

- More generally: functions can be represented by a directed acyclic graph (computation graph)

What are the individual functions f_1, f_2 etc?

Deep Learning:

What function $\mathbf{f}(w, x, y)$ to choose?

- Choose any differentiable composite function

$$\mathbf{f}(w, x, y) = f_1(w_1, y, f_2(w_2, f_3(\ldots f_n(w_n, x)\ldots)))$$

- More generally: functions can be represented by a directed acyclic graph (computation graph)

What are the individual functions f_1, f_2 etc?

- Convolutions

Deep Learning:

What function $\mathbf{f}(w, x, y)$ to choose?

- Choose any differentiable composite function

$$\mathbf{f}(w, x, y) = f_1(w_1, y, f_2(w_2, f_3(\ldots f_n(w_n, x)\ldots)))$$

- More generally: functions can be represented by a directed acyclic graph (computation graph)

What are the individual functions f_1, f_2 etc?

- Convolutions
- Rectified linear units $\max\{0, x\}$

Deep Learning:

What function $\mathbf{f}(w, x, y)$ to choose?

- Choose any differentiable composite function

$$\mathbf{f}(w, x, y) = f_1(w_1, y, f_2(w_2, f_3(\ldots f_n(w_n, x)\ldots)))$$

- More generally: functions can be represented by a directed acyclic graph (computation graph)

What are the individual functions f_1, f_2 etc?

- Convolutions
- Rectified linear units $\max\{0, x\}$
- Maximum-/Average pooling

Deep Learning:

What function $\mathbf{f}(w, x, y)$ to choose?

- Choose any differentiable composite function

$$\mathbf{f}(w, x, y) = f_1(w_1, y, f_2(w_2, f_3(\ldots f_n(w_n, x) \ldots)))$$

- More generally: functions can be represented by a directed acyclic graph (computation graph)

What are the individual functions f_1, f_2 etc?

- Convolutions
- Rectified linear units $\max\{0, x\}$
- Maximum-/Average pooling
- Fully connected layers

Deep Learning:

What function $\mathbf{f}(w, x, y)$ to choose?

- Choose any differentiable composite function

$$\mathbf{f}(w, x, y) = f_1(w_1, y, f_2(w_2, f_3(\ldots f_n(w_n, x)\ldots)))$$

- More generally: functions can be represented by a directed acyclic graph (computation graph)

What are the individual functions f_1, f_2 etc?

- Convolutions
- Rectified linear units $\max\{0, x\}$
- Maximum-/Average pooling
- Fully connected layers
- Soft-max layer

Deep Learning:

What function $\mathbf{f}(w, x, y)$ to choose?

- Choose any differentiable composite function

$$\mathbf{f}(w, x, y) = f_1(w_1, y, f_2(w_2, f_3(\dots f_n(w_n, x)\dots)))$$

- More generally: functions can be represented by a directed acyclic graph (computation graph)

What are the individual functions f_1, f_2 etc?

- Convolutions
- Rectified linear units $\max\{0, x\}$
- Maximum-/Average pooling
- Fully connected layers
- Soft-max layer
- Dropout

Why go deep?

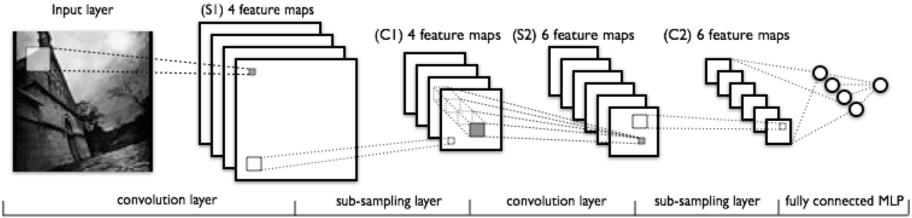
- Formal explanation: Our invited speaker R. Srikant will elaborate some results
- Somewhat formal: Point to Hornik's result and wave hands
- Informal: Adaptive features are extracted from data
- Essentially from a perspective of build gates, e.g. $\sigma(a)$ for the sigmoidal squasher, or $\mathbf{f} = \max(0, x)$ for the RELU

Example function architecture:

LeNet (LeCun 98)

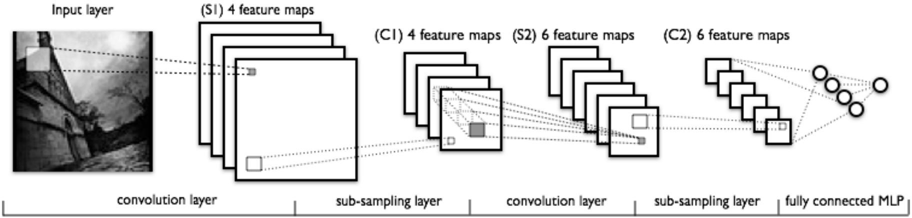
Example function architecture:

LeNet (LeCun 98)



Example function architecture:

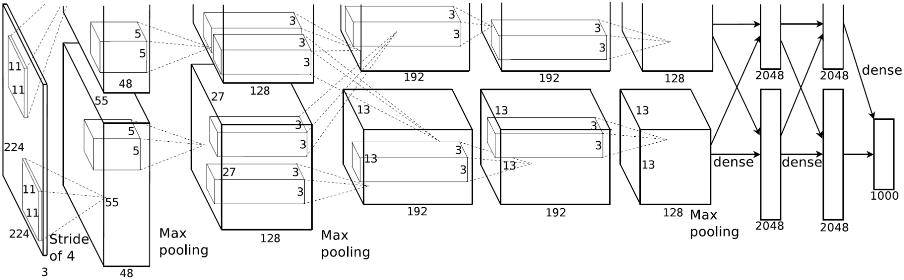
LeNet (LeCun 98)



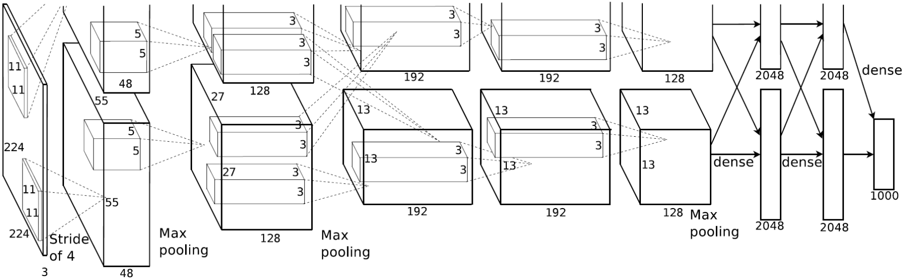
Decreasing spatial resolution and the increasing number of channels

Example function architecture: AlexNet

Example function architecture: AlexNet

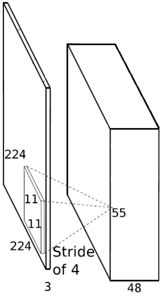


Example function architecture: AlexNet



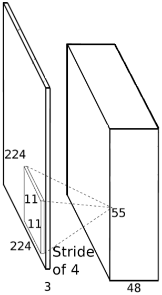
Decreasing spatial resolution and the increasing number of channels

Convolutions:

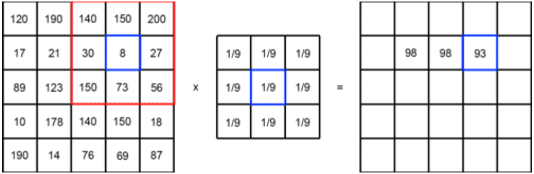


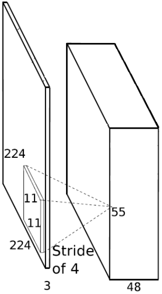
Convolutions:

$$\begin{bmatrix} 120 & 190 & 140 & 150 & 200 \\ 17 & 21 & 30 & 8 & 27 \\ 89 & 123 & 150 & 73 & 56 \\ 10 & 178 & 140 & 150 & 18 \\ 190 & 14 & 76 & 69 & 87 \end{bmatrix} \times \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} = \begin{bmatrix} \; & \; & \; & \; & \; \\ & 98 & 98 & & \\ & & & & \\ & & & & \\ & & & & \end{bmatrix}$$

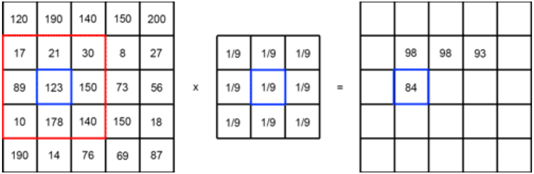


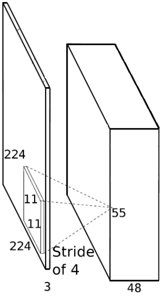
Convolutions:





Convolutions:





Convolutions:

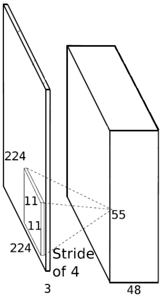
120	190	140	150	200
17	21	30	8	27
89	123	150	73	56
10	178	140	150	18
190	14	76	69	87

x

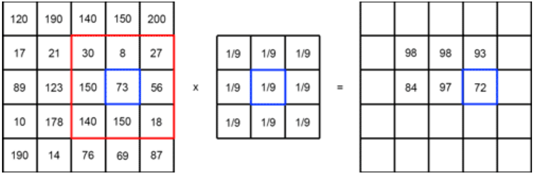
1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

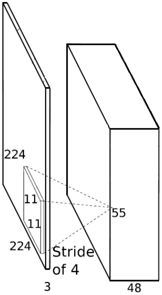
=

	98	98	93	
	84	97		

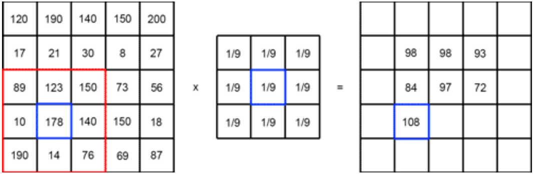


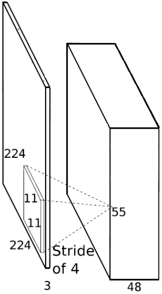
Convolutions:



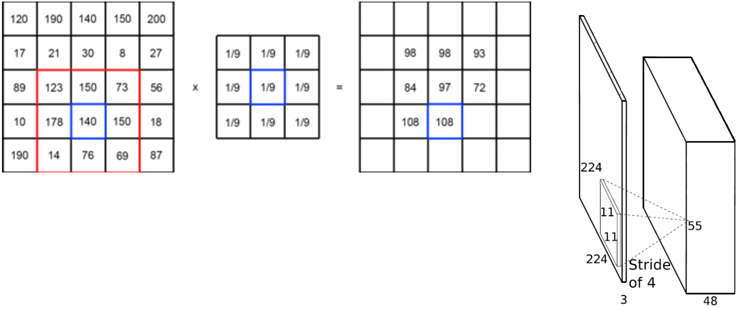


Convolutions:



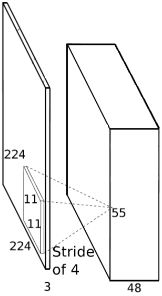


Convolutions:

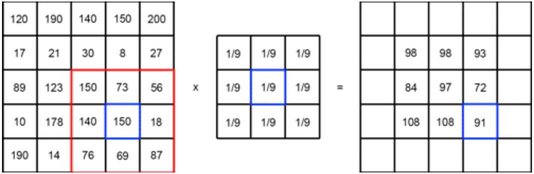


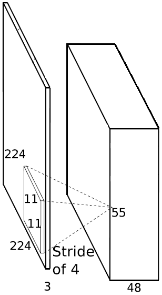
Convolutions:

$$\begin{bmatrix} 120 & 190 & 140 & 150 & 200 \\ 17 & 21 & 30 & 8 & 27 \\ 89 & 123 & 150 & 73 & 56 \\ 10 & 178 & 140 & 150 & 18 \\ 190 & 14 & 76 & 69 & 87 \end{bmatrix} \times \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} = \begin{bmatrix} & & & & \\ & 98 & 98 & 93 & \\ & 84 & 97 & 72 & \\ & 108 & 108 & \boxed{91} & \\ & & & & \end{bmatrix}$$



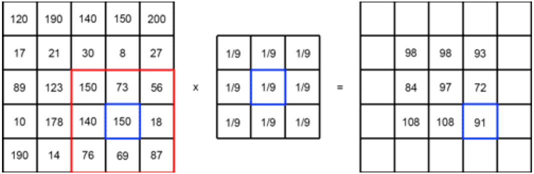
Convolutions:

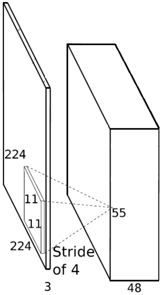




Trainable parameters w:

Convolutions:



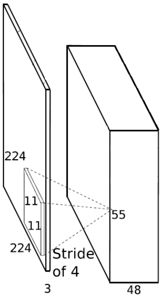


Trainable parameters w:

- Filters (width, height, depth, number)

Convolutions:

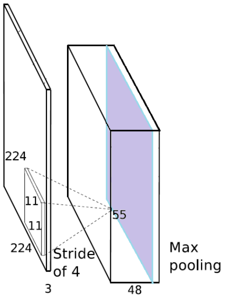
$$\begin{bmatrix} 120 & 190 & 140 & 150 & 200 \\ 17 & 21 & 30 & 8 & 27 \\ 89 & 123 & 150 & 73 & 56 \\ 10 & 178 & 140 & 150 & 18 \\ 190 & 14 & 76 & 69 & 87 \end{bmatrix} \times \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} = \begin{bmatrix} & & & & \\ & 98 & 98 & 93 & \\ & 84 & 97 & 72 & \\ & 108 & 108 & 91 & \\ & & & & \end{bmatrix}$$



Trainable parameters w:

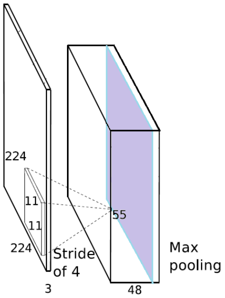
- Filters (width, height, depth, number)
- Bias

Maximum-/Average pooling



Maximum-/Average pooling

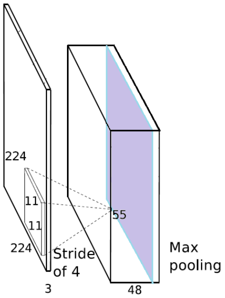
Maximum or average over a spatial region



Maximum-/Average pooling

Maximum or average over a spatial region

Trainable parameters w:

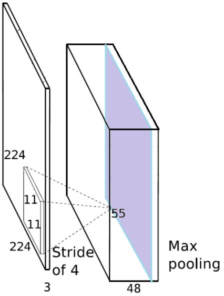


Maximum-/Average pooling

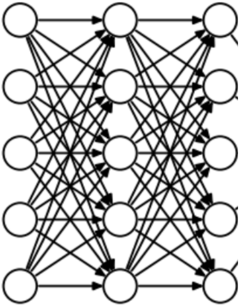
Maximum or average over a spatial region

Trainable parameters w:

- None

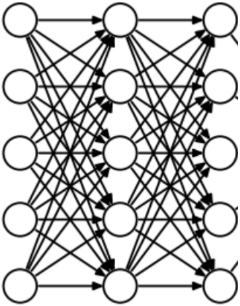


Fully connected layer:



Fully connected layer:

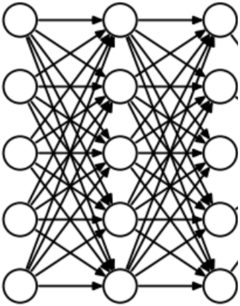
$$Wx + b$$



Fully connected layer:

$$Wx + b$$

Trainable parameters w:

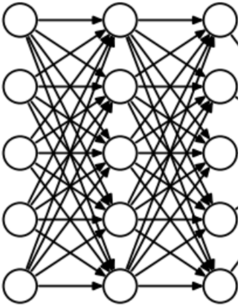


Fully connected layer:

$$Wx + b$$

Trainable parameters w:

- Matrix

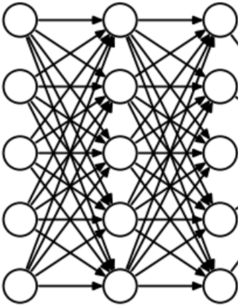


Fully connected layer:

$$Wx + b$$

Trainable parameters w:

- Matrix
- Bias



Soft-max layer:

Soft-max layer:

$$x \quad \longrightarrow \quad \frac{\exp x_i}{\sum_j \exp x_j}$$

Soft-max layer:

$$x \quad \longrightarrow \quad \frac{\exp x_i}{\sum_j \exp x_j}$$

Trainable parameters w:

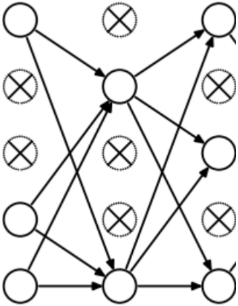
Soft-max layer:

$$x \quad \longrightarrow \quad \frac{\exp x_i}{\sum_j \exp x_j}$$

Trainable parameters w:

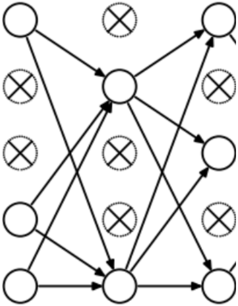
- None

Dropout layer:



Dropout layer:

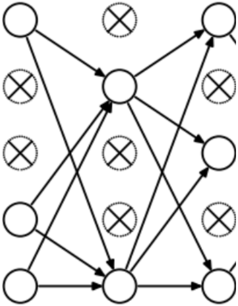
Randomly drop some activations



Dropout layer:

Randomly drop some activations

Trainable parameters w:

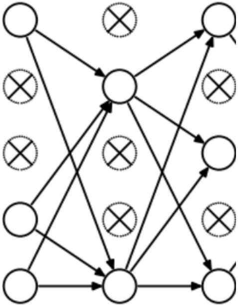


Dropout layer:

Randomly drop some activations

Trainable parameters w:

- None



Deep net training:

$$\min_{w} \frac{C}{2}\|w\|_2^2 + \sum_{i\in\mathcal{D}} \ln \sum_{\hat{y}} \exp F(w, x, \hat{y}) - F(w, x, y^{(i)})$$

Deep net training:

$$\min_w \frac{C}{2}\|w\|_2^2 + \sum_{i\in\mathcal{D}} \ln \sum_{\hat{y}} \exp F(w,x,\hat{y}) - F(w,x,y^{(i)})$$

Often also referred to as maximizing the regularized cross entropy:

Deep net training:

$$\min_w \frac{C}{2}\|w\|_2^2 + \sum_{i\in\mathcal{D}} \ln \sum_{\hat{y}} \exp F(w,x,\hat{y}) - F(w,x,y^{(i)})$$

Often also referred to as maximizing the regularized cross entropy:

$$\max_w -\frac{C}{2}\|w\|_2^2 + \sum_{i\in\mathcal{D}} \sum_{\hat{y}} p_{\text{GT}}^{(i)}(\hat{y}) \ln p(\hat{y}|x) \quad \text{with}$$

Deep net training:

$$\min_w \frac{C}{2}\|w\|_2^2 + \sum_{i\in\mathcal{D}} \ln \sum_{\hat{y}} \exp F(w,x,\hat{y}) - F(w,x,y^{(i)})$$

Often also referred to as maximizing the regularized cross entropy:

$$\max_w -\frac{C}{2}\|w\|_2^2 + \sum_{i\in\mathcal{D}} \sum_{\hat{y}} p_{\text{GT}}^{(i)}(\hat{y}) \ln p(\hat{y}|x) \quad \text{with} \left\{ \begin{array}{ccc} p_{\text{GT}}^{(i)}(\hat{y}) & = & \delta(\hat{y} = y^{(i)}) \\ p(\hat{y}|x) & \propto & \exp F(w,x,\hat{y}) \end{array} \right.$$

Deep net training:

$$\min_w \frac{C}{2}\|w\|_2^2 + \sum_{i\in\mathcal{D}} \ln \sum_{\hat{y}} \exp F(w,x,\hat{y}) - F(w,x,y^{(i)})$$

Often also referred to as maximizing the regularized cross entropy:

$$\max_w -\frac{C}{2}\|w\|_2^2 + \sum_{i\in\mathcal{D}} \sum_{\hat{y}} p_{\text{GT}}^{(i)}(\hat{y}) \ln p(\hat{y}|x) \quad \text{with} \left\{ \begin{array}{ccc} p_{\text{GT}}^{(i)}(\hat{y}) & = & \delta(\hat{y} = y^{(i)}) \\ p(\hat{y}|x) & \propto & \exp F(w,x,\hat{y}) \end{array} \right.$$

What is C?

Deep net training:

$$\min_w \frac{C}{2}\|w\|_2^2 + \sum_{i \in \mathcal{D}} \ln \sum_{\hat{y}} \exp F(w, x, \hat{y}) - F(w, x, y^{(i)})$$

Often also referred to as maximizing the regularized cross entropy:

$$\max_w -\frac{C}{2}\|w\|_2^2 + \sum_{i \in \mathcal{D}} \sum_{\hat{y}} p_{\mathrm{GT}}^{(i)}(\hat{y}) \ln p(\hat{y}|x) \quad \text{with} \left\{ \begin{array}{ccc} p_{\mathrm{GT}}^{(i)}(\hat{y}) & = & \delta(\hat{y} = y^{(i)}) \\ p(\hat{y}|x) & \propto & \exp F(w, x, \hat{y}) \end{array} \right.$$

What is C? Weight decay (aka regularization constant)

Deep net training:

$$\min_w \frac{C}{2}\|w\|_2^2 + \sum_{i\in\mathcal{D}} \ln \sum_{\hat{y}} \exp F(w,x,\hat{y}) - F(w,x,y^{(i)})$$

Often also referred to as maximizing the regularized cross entropy:

$$\max_w -\frac{C}{2}\|w\|_2^2 + \sum_{i\in\mathcal{D}} \sum_{\hat{y}} p_{\text{GT}}^{(i)}(\hat{y}) \ln p(\hat{y}|x) \quad \text{with} \left\{ \begin{array}{ccc} p_{\text{GT}}^{(i)}(\hat{y}) & = & \delta(\hat{y} = y^{(i)}) \\ p(\hat{y}|x) & \propto & \exp F(w,x,\hat{y}) \end{array} \right.$$

What is C? Weight decay (aka regularization constant)

How to optimize this?

Deep net training:

$$\min_w \frac{C}{2}\|w\|_2^2 + \sum_{i\in\mathcal{D}} \ln \sum_{\hat{y}} \exp F(w,x,\hat{y}) - F(w,x,y^{(i)})$$

Often also referred to as maximizing the regularized cross entropy:

$$\max_w -\frac{C}{2}\|w\|_2^2 + \sum_{i\in\mathcal{D}} \sum_{\hat{y}} p_{\text{GT}}^{(i)}(\hat{y}) \ln p(\hat{y}|x) \quad \text{with} \left\{ \begin{array}{lcl} p_{\text{GT}}^{(i)}(\hat{y}) & = & \delta(\hat{y} = y^{(i)}) \\ p(\hat{y}|x) & \propto & \exp F(w,x,\hat{y}) \end{array} \right.$$

What is C? Weight decay (aka regularization constant)

How to optimize this?

Stochastic gradient descent with momentum: What was this again?

Important remark:

Important remark:

Since $F(w, x, y)$ is no longer constrained in any form, the loss function is generally no longer convex.

Important remark:

Since $F(w, x, y)$ is no longer constrained in any form, the loss function is generally no longer convex.

Implications:

Important remark:

Since $F(w, x, y)$ is no longer constrained in any form, the loss function is generally no longer convex.

Implications:

- We are no longer guaranteed to find the global optimum

Important remark:

Since $F(w, x, y)$ is no longer constrained in any form, the loss function is generally no longer convex.

Implications:

- We are no longer guaranteed to find the global optimum
- Initialization of w matters

Gradient of

$$\min_w \frac{C}{2}\|w\|_2^2 + \sum_{i\in\mathcal{D}} \ln \sum_{\hat{y}} \exp F(w, x, \hat{y}) - F(w, x, y^{(i)})$$

is?

Gradient of

$$\min_w \frac{C}{2}\|w\|_2^2 + \sum_{i\in\mathcal{D}} \ln \sum_{\hat{y}} \exp F(w,x,\hat{y}) - F(w,x,y^{(i)})$$

is?

$$Cw + \sum_{i\in\mathcal{D}} \sum_{\hat{y}} \left(p(\hat{y}|x) - \delta(\hat{y} = y^{(i)})\right) \frac{\partial F(w,x,\hat{y})}{\partial w}$$

Gradient of

$$\min_{w} \frac{C}{2}\|w\|_2^2 + \sum_{i\in\mathcal{D}} \ln \sum_{\hat{y}} \exp F(w, x, \hat{y}) - F(w, x, y^{(i)})$$

is?

$$Cw + \sum_{i\in\mathcal{D}} \sum_{\hat{y}} \left(p(\hat{y}|x) - \delta(\hat{y} = y^{(i)}) \right) \frac{\partial F(w, x, \hat{y})}{\partial w}$$

How to compute this:

Gradient of

$$\min_w \frac{C}{2}\|w\|_2^2 + \sum_{i \in \mathcal{D}} \ln \sum_{\hat{y}} \exp F(w, x, \hat{y}) - F(w, x, y^{(i)})$$

is?

$$Cw + \sum_{i \in \mathcal{D}} \sum_{\hat{y}} \left(p(\hat{y}|x) - \delta(\hat{y} = y^{(i)}) \right) \frac{\partial F(w, x, \hat{y})}{\partial w}$$

How to compute this:

- $p(\hat{y}|x) = \frac{\exp F(w,x,\hat{y})}{\sum_{\tilde{y}} \exp F(w,x,\tilde{y})}$

Gradient of

$$\min_w \frac{C}{2}\|w\|_2^2 + \sum_{i \in \mathcal{D}} \ln \sum_{\hat{y}} \exp F(w, x, \hat{y}) - F(w, x, y^{(i)})$$

is?

$$Cw + \sum_{i \in \mathcal{D}} \sum_{\hat{y}} \left(p(\hat{y}|x) - \delta(\hat{y} = y^{(i)})\right) \frac{\partial F(w, x, \hat{y})}{\partial w}$$

How to compute this:

- $p(\hat{y}|x) = \frac{\exp F(w,x,\hat{y})}{\sum_{\tilde{y}} \exp F(w,x,\tilde{y})}$ via soft-max which takes F as input

Gradient of

$$\min_w \frac{C}{2}\|w\|_2^2 + \sum_{i\in\mathcal{D}} \ln \sum_{\hat{y}} \exp F(w, x, \hat{y}) - F(w, x, y^{(i)})$$

is?

$$Cw + \sum_{i\in\mathcal{D}} \sum_{\hat{y}} \left(p(\hat{y}|x) - \delta(\hat{y} = y^{(i)})\right) \frac{\partial F(w, x, \hat{y})}{\partial w}$$

How to compute this:

- $p(\hat{y}|x) = \frac{\exp F(w,x,\hat{y})}{\sum_{\tilde{y}} \exp F(w,x,\tilde{y})}$ via soft-max which takes F as input
  - A dedicated cross-entropy layer exists (numerically more stable)

Gradient of

$$\min_{w} \frac{C}{2}\|w\|_2^2 + \sum_{i\in\mathcal{D}} \ln \sum_{\hat{y}} \exp F(w,x,\hat{y}) - F(w,x,y^{(i)})$$

is?

$$Cw + \sum_{i\in\mathcal{D}} \sum_{\hat{y}} \left(p(\hat{y}|x) - \delta(\hat{y} = y^{(i)})\right) \frac{\partial F(w,x,\hat{y})}{\partial w}$$

How to compute this:

- $p(\hat{y}|x) = \frac{\exp F(w,x,\hat{y})}{\sum_{\tilde{y}} \exp F(w,x,\tilde{y})}$ via soft-max which takes F as input
  - A dedicated cross-entropy layer exists (numerically more stable)
- $\frac{\partial F(w,x,\hat{y})}{\partial w}$ via backpropagation

Composite function: Backpropagation example

Composite function: Backpropagation example

$$F(w,x,y) = f_1(w_1, f_2(w_2, f_3(w_3, x))) \text{ with activations } \left\{ \begin{array}{lcl} x_2 & = & f_3(w_3, x) \\ x_1 & = & f_2(w_2, x_2) \end{array} \right.$$

Composite function: Backpropagation example

$$F(w,x,y) = f_1(w_1, f_2(w_2, f_3(w_3, x))) \text{ with activations } \begin{cases} x_2 & = & f_3(w_3, x) \\ x_1 & = & f_2(w_2, x_2) \end{cases}$$

What is $\frac{\partial F(w,x,y)}{\partial w_3}$?

Composite function: Backpropagation example

$$F(w,x,y) = f_1(w_1, f_2(w_2, f_3(w_3, x))) \text{ with activations } \left\{ \begin{array}{lcl} x_2 & = & f_3(w_3, x) \\ x_1 & = & f_2(w_2, x_2) \end{array} \right.$$

What is $\frac{\partial F(w,x,y)}{\partial w_3}$?

$$\frac{\partial f_1}{\partial x_1} \cdot \frac{\partial x_1}{\partial x_2} \cdot \frac{\partial x_2}{\partial w_3} =$$

Composite function: Backpropagation example

$$F(w,x,y) = f_1(w_1, f_2(w_2, f_3(w_3, x))) \text{ with activations } \begin{cases} x_2 & = & f_3(w_3, x) \\ x_1 & = & f_2(w_2, x_2) \end{cases}$$

What is $\frac{\partial F(w,x,y)}{\partial w_3}$?

$$\frac{\partial f_1}{\partial x_1} \cdot \frac{\partial x_1}{\partial x_2} \cdot \frac{\partial x_2}{\partial w_3} = \frac{\partial f_1}{\partial f_2} \cdot \frac{\partial f_2}{\partial f_3} \cdot \frac{\partial f_3}{\partial w_3}$$

Composite function: Backpropagation example

$$F(w,x,y) = f_1(w_1, f_2(w_2, f_3(w_3, x))) \text{ with activations } \begin{cases} x_2 & = & f_3(w_3, x) \\ x_1 & = & f_2(w_2, x_2) \end{cases}$$

What is $\frac{\partial F(w,x,y)}{\partial w_3}$?

$$\frac{\partial f_1}{\partial x_1} \cdot \frac{\partial x_1}{\partial x_2} \cdot \frac{\partial x_2}{\partial w_3} = \frac{\partial f_1}{\partial f_2} \cdot \frac{\partial f_2}{\partial f_3} \cdot \frac{\partial f_3}{\partial w_3}$$

What is $\frac{\partial F(w,x,y)}{\partial w_2}$?

Composite function: Backpropagation example

$$F(w, x, y) = f_1(w_1, f_2(w_2, f_3(w_3, x))) \text{ with activations } \begin{cases} x_2 &= f_3(w_3, x) \\ x_1 &= f_2(w_2, x_2) \end{cases}$$

What is $\frac{\partial F(w,x,y)}{\partial w_3}$?

$$\frac{\partial f_1}{\partial x_1} \cdot \frac{\partial x_1}{\partial x_2} \cdot \frac{\partial x_2}{\partial w_3} = \frac{\partial f_1}{\partial f_2} \cdot \frac{\partial f_2}{\partial f_3} \cdot \frac{\partial f_3}{\partial w_3}$$

What is $\frac{\partial F(w,x,y)}{\partial w_2}$?

$$\frac{\partial f_1}{\partial x_1} \cdot \frac{\partial x_1}{\partial w_2} =$$

Composite function: Backpropagation example

$$F(w,x,y) = f_1(w_1, f_2(w_2, f_3(w_3, x))) \text{ with activations } \begin{cases} x_2 &= f_3(w_3, x) \\ x_1 &= f_2(w_2, x_2) \end{cases}$$

What is $\frac{\partial F(w,x,y)}{\partial w_3}$?

$$\frac{\partial f_1}{\partial x_1} \cdot \frac{\partial x_1}{\partial x_2} \cdot \frac{\partial x_2}{\partial w_3} = \frac{\partial f_1}{\partial f_2} \cdot \frac{\partial f_2}{\partial f_3} \cdot \frac{\partial f_3}{\partial w_3}$$

What is $\frac{\partial F(w,x,y)}{\partial w_2}$?

$$\frac{\partial f_1}{\partial x_1} \cdot \frac{\partial x_1}{\partial w_2} = \frac{\partial f_1}{\partial f_2} \cdot \frac{\partial f_2}{\partial w_2}$$

Composite function: Backpropagation example

$$F(w,x,y) = f_1(w_1, f_2(w_2, f_3(w_3, x))) \text{ with activations } \begin{cases} x_2 &= f_3(w_3, x) \\ x_1 &= f_2(w_2, x_2) \end{cases}$$

What is $\frac{\partial F(w,x,y)}{\partial w_3}$?

$$\frac{\partial f_1}{\partial x_1} \cdot \frac{\partial x_1}{\partial x_2} \cdot \frac{\partial x_2}{\partial w_3} = \frac{\partial f_1}{\partial f_2} \cdot \frac{\partial f_2}{\partial f_3} \cdot \frac{\partial f_3}{\partial w_3}$$

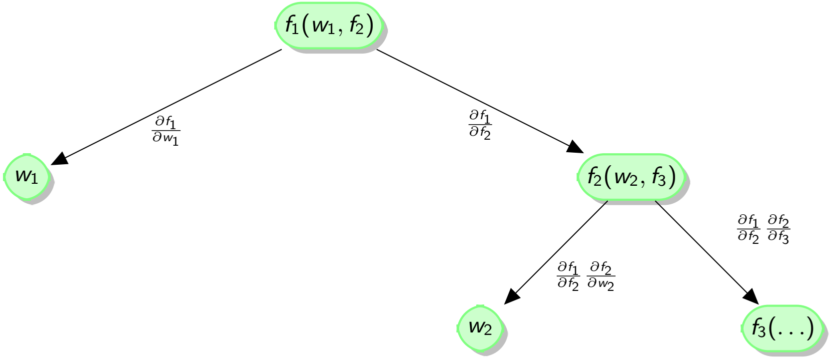
What is $\frac{\partial F(w,x,y)}{\partial w_2}$?

$$\frac{\partial f_1}{\partial x_1} \cdot \frac{\partial x_1}{\partial w_2} = \frac{\partial f_1}{\partial f_2} \cdot \frac{\partial f_2}{\partial w_2}$$

Generally: To avoid repeated computation, backpropagation on a directed acyclic graph.

Composite function represented as a directed a-cyclic graph

$$\ell(x, w) = f_1(w_1, f_2(w_2, f_3(\ldots)))$$



Repeated application of chain rule for efficient computation of all gradients

Note:

Note:

A deep net with a single fully connected layer is equivalent to logistic regression

Note:

A deep net with a single fully connected layer is equivalent to logistic regression

Advantages of deep nets compared to usage of hand-crafted features:

Note:

A deep net with a single fully connected layer is equivalent to logistic regression

Advantages of deep nets compared to usage of hand-crafted features:

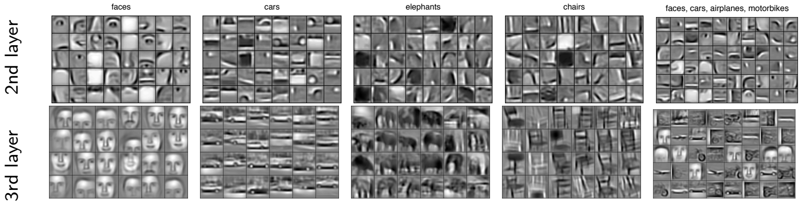
Deep nets automatically learn feature space transformations (hierarchical abstractions of data) such that data is easily separable at the output

Note:

A deep net with a single fully connected layer is equivalent to logistic regression

Advantages of deep nets compared to usage of hand-crafted features:

Deep nets automatically learn feature space transformations (hierarchical abstractions of data) such that data is easily separable at the output

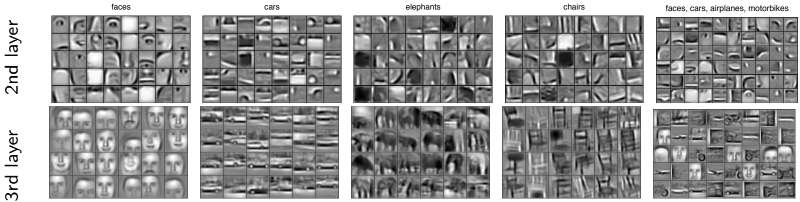


Note:

A deep net with a single fully connected layer is equivalent to logistic regression

Advantages of deep nets compared to usage of hand-crafted features:

Deep nets automatically learn feature space transformations (hierarchical abstractions of data) such that data is easily separable at the output



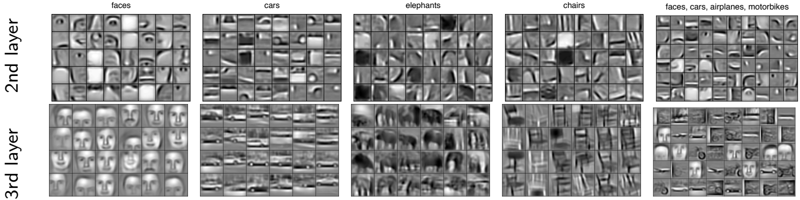
Disadvantage of deep nets compared to usage of features:

Note:

A deep net with a single fully connected layer is equivalent to logistic regression

Advantages of deep nets compared to usage of hand-crafted features:

Deep nets automatically learn feature space transformations (hierarchical abstractions of data) such that data is easily separable at the output



Disadvantage of deep nets compared to usage of features:

Deep nets are computationally demanding (GPUs) and require significant amounts of training data

Why this recent popularity:

Why this recent popularity:

- Sufficient computational resources

Why this recent popularity:

- Sufficient computational resources
- Sufficient data

Why this recent popularity:

- Sufficient computational resources
- Sufficient data
- Sufficient algorithmic advances

Why this recent popularity:

- Sufficient computational resources
- Sufficient data
- Sufficient algorithmic advances

This combination lead to significant performance improvements on many datasets

Algorithmic advances:

Algorithmic advances:

- Rectified linear unit ($\max\{0, x\}$) activation as opposed to sigmoid

Algorithmic advances:

- Rectified linear unit ($\max\{0, x\}$) activation as opposed to sigmoid
  - Fixed the vanishing gradient problem for lower layers close to the input

Algorithmic advances:

- Rectified linear unit ($\max\{0, x\}$) activation as opposed to sigmoid
  - Fixed the vanishing gradient problem for lower layers close to the input
- Dropout

Algorithmic advances:

- Rectified linear unit ($\max\{0, x\}$) activation as opposed to sigmoid
  - Fixed the vanishing gradient problem for lower layers close to the input
- Dropout
  - Decorrelates different units, i.e., they learn different features

Algorithmic advances:

- Rectified linear unit ($\max\{0, x\}$) activation as opposed to sigmoid
  - Fixed the vanishing gradient problem for lower layers close to the input
- Dropout
  - Decorrelates different units, i.e., they learn different features
- Good initialization heuristics

Algorithmic advances:

- Rectified linear unit ($\max\{0, x\}$) activation as opposed to sigmoid
  - Fixed the vanishing gradient problem for lower layers close to the input
- Dropout
  - Decorrelates different units, i.e., they learn different features
- Good initialization heuristics
  - Less prone to getting stuck in bad local optima

Algorithmic advances:

- Rectified linear unit ($\max\{0, x\}$) activation as opposed to sigmoid
  - Fixed the vanishing gradient problem for lower layers close to the input
- Dropout
  - Decorrelates different units, i.e., they learn different features
- Good initialization heuristics
  - Less prone to getting stuck in bad local optima
- Batch-Normalization during training

Algorithmic advances:

- Rectified linear unit ($\max\{0, x\}$) activation as opposed to sigmoid
  - Fixed the vanishing gradient problem for lower layers close to the input
- Dropout
  - Decorrelates different units, i.e., they learn different features
- Good initialization heuristics
  - Less prone to getting stuck in bad local optima
- Batch-Normalization during training
  - Fixes covariate shift when training really deep nets

Algorithmic advances:

- Rectified linear unit ($\max\{0, x\}$) activation as opposed to sigmoid
  - Fixed the vanishing gradient problem for lower layers close to the input
- Dropout
  - Decorrelates different units, i.e., they learn different features
- Good initialization heuristics
  - Less prone to getting stuck in bad local optima
- Batch-Normalization during training
  - Fixes covariate shift when training really deep nets
  - Normalize by subtracting mean and dividing by standard deviation

Popular architectures:

Popular architectures:

- LeNet

Popular architectures:

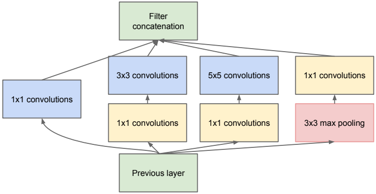
- LeNet
- AlexNet

Popular architectures:

- LeNet
- AlexNet
- VGG (16/19 layers, mostly 3x3 convolutions)

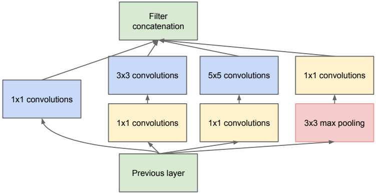
Popular architectures:

- LeNet
- AlexNet
- VGG (16/19 layers, mostly 3x3 convolutions)
- GoogLeNet (inception module)

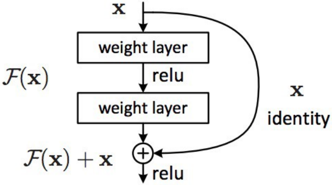


Popular architectures:

- LeNet
- AlexNet
- VGG (16/19 layers, mostly 3x3 convolutions)
- GoogLeNet (inception module)



- ResNet (residual connections)



Imagenet Challenge:

Imagenet Challenge:

- A large dataset: 1.2M images, 1000 categories

Imagenet Challenge:

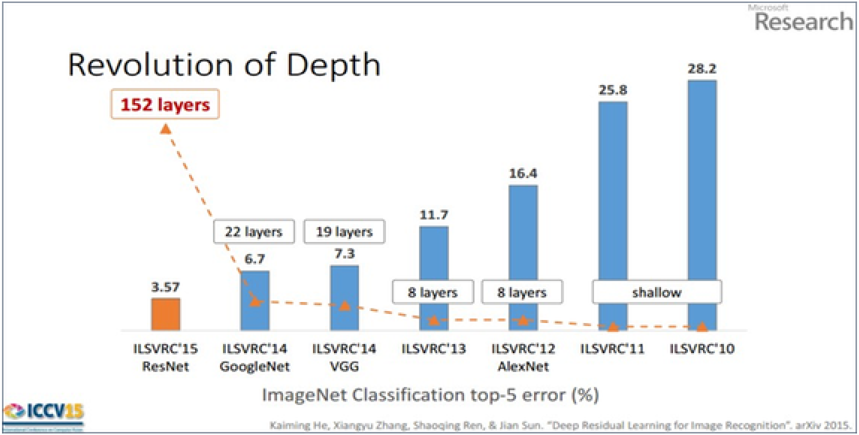
- A large dataset: 1.2M images, 1000 categories
- AlexNet was run on the GPU, i.e., sufficient computational resources

Imagenet Challenge:

- A large dataset: 1.2M images, 1000 categories
- AlexNet was run on the GPU, i.e., sufficient computational resources
- Rectified linear units rather than sigmoid units simplify optimization

Results:

Results:



Microsoft
Research
Revolution of Depth
152 layers
22 layers
19 layers
8 layers
8 layers
shallow
3.57
6.7
7.3
11.7
16.4
25.8
28.2
ILSVRC'15
ResNet
ILSVRC'14
GoogleNet
ILSVRC'14
VGG
ILSVRC'13
ILSVRC'12
AlexNet
ILSVRC'11
ILSVRC'10
ImageNet Classification top-5 error (%)
ICCV15
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

[1] Trevor Campbell, Sameera S. Ponda, Girish Chowdhary, and Jonathan P. How.
Planning under uncertainty using nonparametric bayesian models.
In *AIAA Guidance, Navigation, and Control Conference (GNC)*, August 2012.

[2] G. Chowdhary, H.A. Kingravi, J.P. How, and P.A. Vela.
Bayesian nonparametric adaptive control using gaussian processes.
Neural Networks and Learning Systems, IEEE Transactions on, PP(99):1–1, 2014.

[3] Lehel Csatö and Manfred Opper.
Sparse on-line gaussian processes.
Neural computation, 14(3):641–668, 2002.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville.
Deep Learning.
MIT Press, 2016.
http://www.deeplearningbook.org.

[5] K. Hornik, M. Stinchcombe, and H. White.
Multilayer feedforward networks are universal approximators.
Neural Networks, 2:359–366, 1989.

[6] K. Murphy.
Adaptive Computation and Machine Learning: Machine Learning: A Probabilistic Perspective.
MIT Press, Cambridge, MA, 2012.

[7] C Paciorek and M Schervish.
Nonstationary covariance functions for gaussian process regression.
Advances in neural information processing systems, 16:273–280, 2004.

[8] J. Park and I.W. Sandberg.
Universal approximation using radial-basis-function networks.
Neural Computatations, 3:246–257, 1991.

[9] C.E. Rasmussen and C.K.I. Williams.
Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning).
The MIT Press, 2005.

References II

[10] Yee Whye Teh.
A hierarchical bayesian language model based on pitman-yor processes.
In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 985–992. Association for Computational Linguistics, 2006.