Generalization

Need for Inductive Bias

Understanding Machine Learning

Summary

generalization as core of ML:

empirical risk minimization (training error) as proxy for minimizing unknown population risk (test error, aka generalization error or out-of-sample error)

generalization gap: difference between test and training error

- **interpolation** to unencountered samples from training environment
- **extrapolation** to testing conditions differing from training environment (aka out-of-distribution)

curse of dimensionality: many features (dimensions) → lots of data needed to densely sample volume

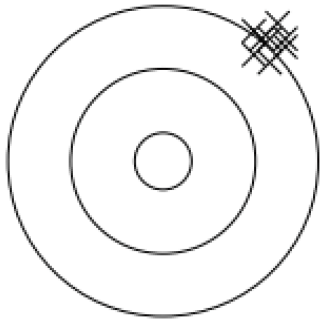
but reality is friendly: most high-dimensional data sets reside on lower-dimensional manifolds (manifold hypothesis) → enabling effectiveness of ML

→ need for appropriate **inductive bias** (different forms: model design, regularization, …)

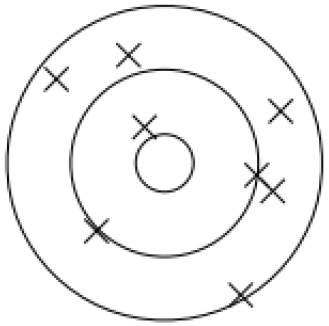
Bias-Variance Tradeoff

Bias, Variance, Irreducible Error

think of fitting ML algorithms as repeatable processes with different (i.i.d.) data sets



bias:
due to too simplistic model
(same for all training data sets)
“underfitting”



variance:
due to sensitivity to specifics (noise)
of different training data sets
“overfitting”

irreducible error (aka Bayes error):

inherent randomness (target generated from random variable following probability distribution)

→limiting accuracy of ideal model

different potential reasons for inherent randomness (noise): complexity, missing information, …

Mean Squared Error

fit function $\hat{f}(\boldsymbol{x})$ to training data set D to approximate assumed true $f(\boldsymbol{x})$

$$y_i = f\left(x_{i1}, x_{i2}, \cdots, x_{ip}\right) + \varepsilon_i$$

- with $\boldsymbol{x} = (x_1, x_2, \cdots, x_p)$
- $D = \{(y_1, \boldsymbol{x}_1), \cdots, (y_n, \boldsymbol{x}_n)\}$
- noise ε: e.g., from Gaussian with zero mean and variance σ^2 (homoscedasticity)

measure mean squared error MSE

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{f}(\boldsymbol{x}_i; D)\right)^2$$

for training data set D (in-sample)

$$\mathrm{MSE} = \frac{1}{q}\sum_{i=n+1}^{n+q}\left(y_i - \hat{f}(\boldsymbol{x}_i; D)\right)^2$$

and test data set T (out-of-sample)

$$T = \left\{(y_{n+1}, \boldsymbol{x}_{n+1}), \cdots, (y_{n+q}, \boldsymbol{x}_{n+q})\right\}$$

Bias-Variance Decomposition

expected squared error on individual sample in test data set T (average over T for MSE):

$$E_{D,\varepsilon}\left[\left(y_i - \hat{f}(\boldsymbol{x}_i; D)\right)^2\right]$$

$$= \left(E_D\left[\hat{f}(\boldsymbol{x}_i; D)\right] - f(\boldsymbol{x}_i)\right)^2 + E_D\left[\left(E_D\left[\hat{f}(\boldsymbol{x}_i; D)\right] - \hat{f}(\boldsymbol{x}_i; D)\right)^2\right] + \sigma^2$$

bias variance irreducible error

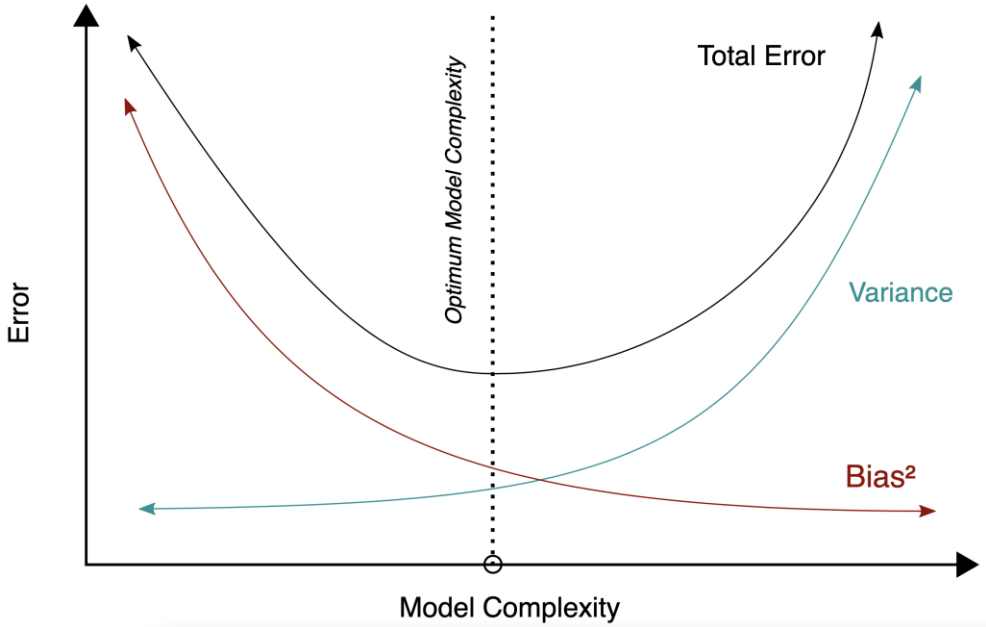
meaning of E_D:
"averaging" predictions from models trained on different choices of D
(independently sampled from same $p(y, \boldsymbol{x})$)

Bias-Variance Tradeoff

- fundamental concept in classical statistical learning theory
- models of higher complexity have lower bias but higher variance (given the same number of training examples)

- generalization error follows U-shaped curve: overfitting once model complexity (number of parameters) passes certain threshold
- overfitting: variance term dominating test error

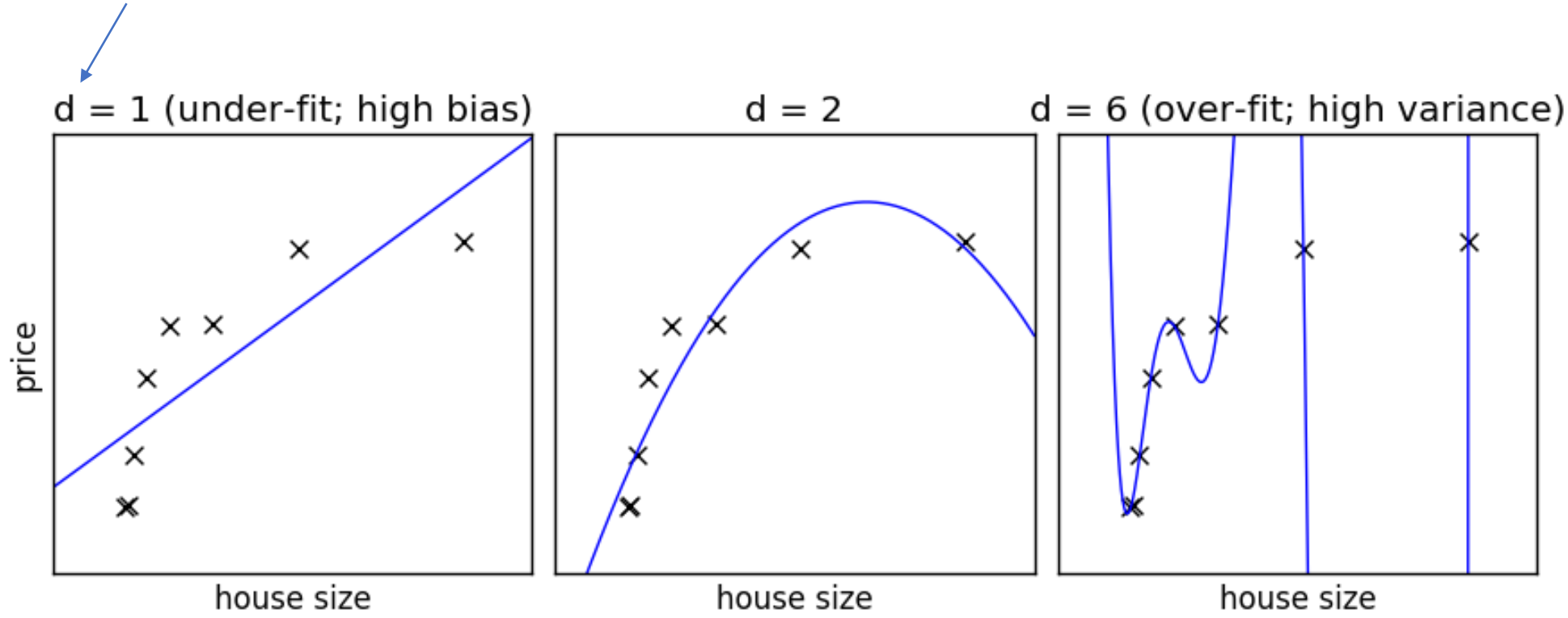
→ increasing model complexity increases test error



from wikipedia

Example: Non-Linear Function Approximation

degree of fitted polynomial



from scikit-learn documentation

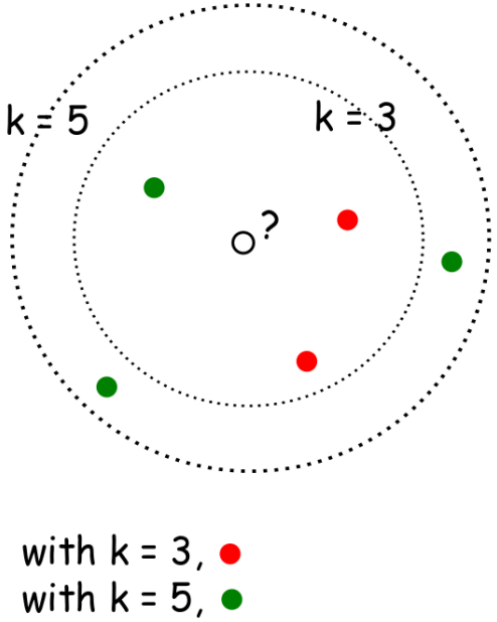
Example: k-Nearest Neighbors

- local method, instance-based learning
- non-parametric
- distance defined by metric on $\boldsymbol{x}$ (e.g., Euclidean)

regression:

$$\hat{f}(\boldsymbol{x}_0) = \frac{1}{k}\sum_{j=1}^{k} y_j$$

with j running over k nearest neighbors of $\boldsymbol{x}_0$



- low k: low bias but high variance
- high k: low variance but high bias

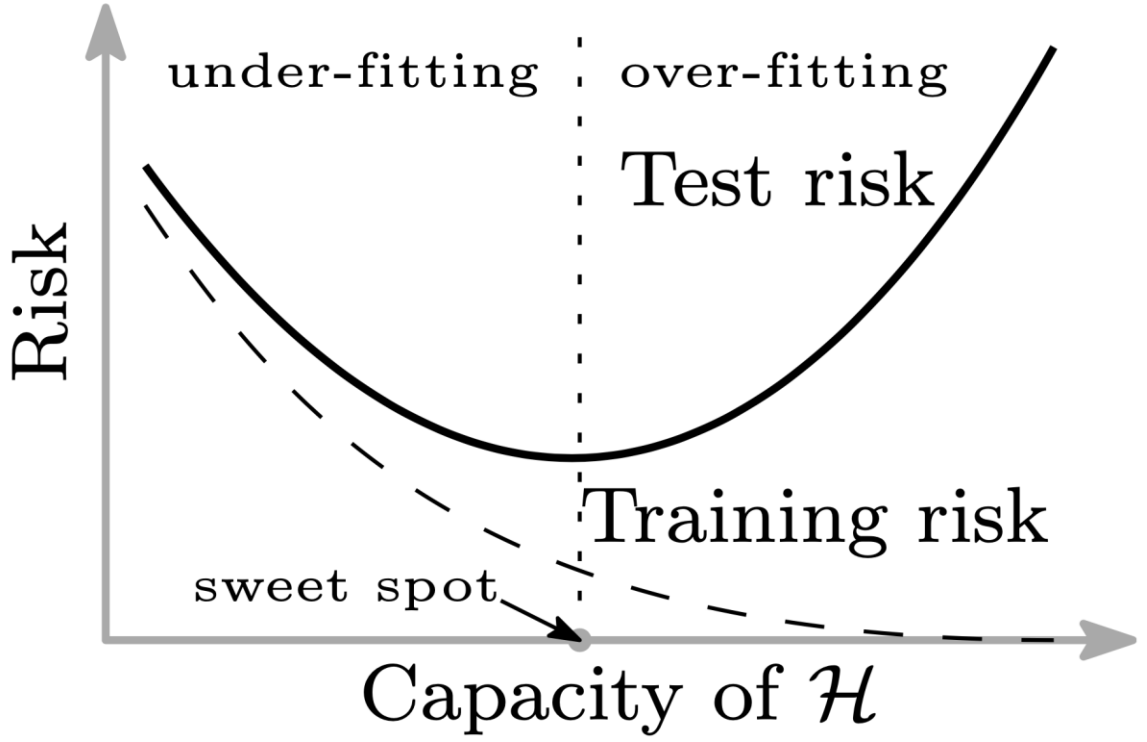
$$bias = f(\boldsymbol{x}) - \frac{1}{k}\sum_{j=1}^{k} y_j$$

$$var = \frac{\sigma^2}{k}$$

Problem of Finding Complexity Sweet Spot

- training/in-sample error keeps decreasing with more complex model (less bias)
- test/out-of-sample error not easily accessible during training

→ e.g., cross-validation, early stopping



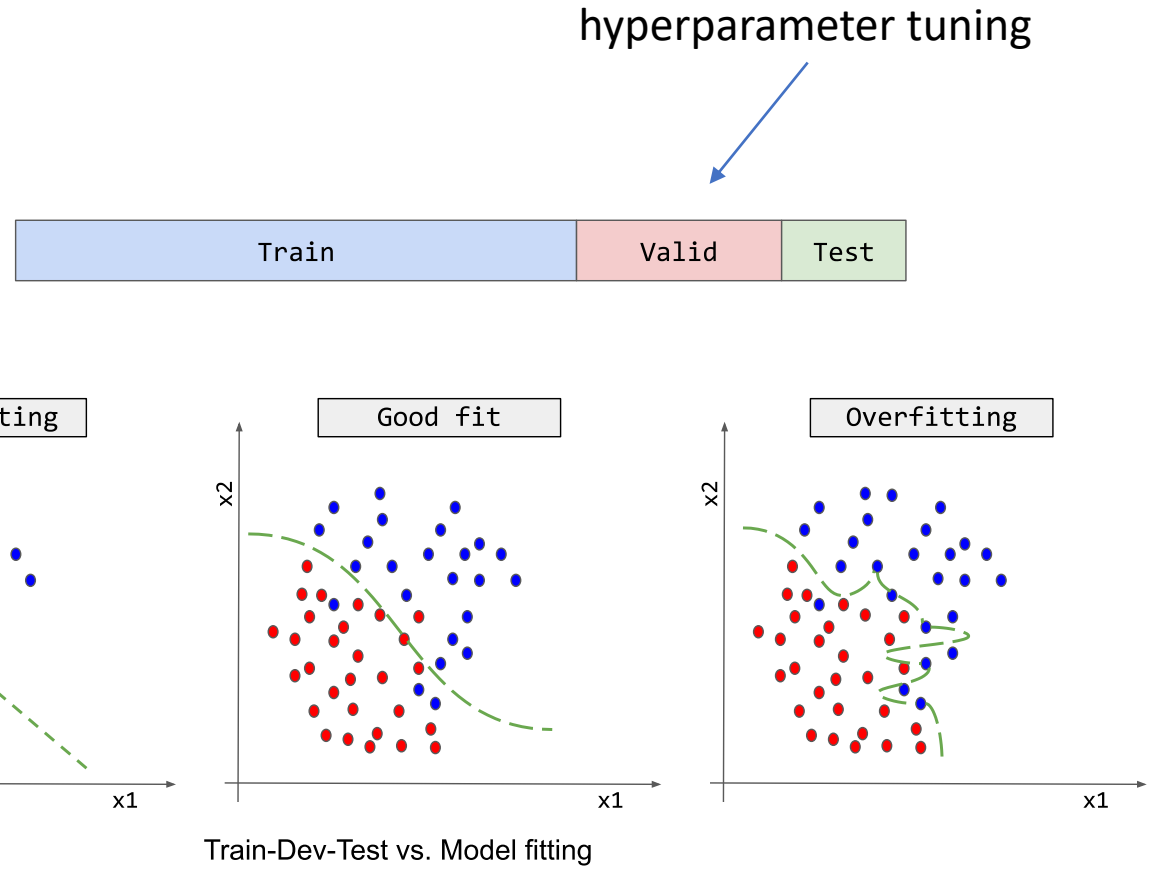
source

Hyperparameters

model complexity often controlled via hyperparameters (parameters not fitted but set in advance)

examples:

- degree of fitted polynomial d
- number of considered nearest neighbors k
- maximum depth of decision tree
- number of trees in random forest
- …



Train-Dev-Test vs. Model fitting

Methods for …

regularization

reducing bias:

- kNN: consider less neighbors
- decision tree: greater depth
- ensemble: boosting
- neural network: more hidden nodes
- more model features
- larger training data set

reducing variance:

- kNN: consider more neighbors
- decision tree: shallower depth
- ensemble: bagging (kind of regularization)
- neural network: less hidden nodes
- dimensionality reduction, feature selection
- larger training data set

Regularization

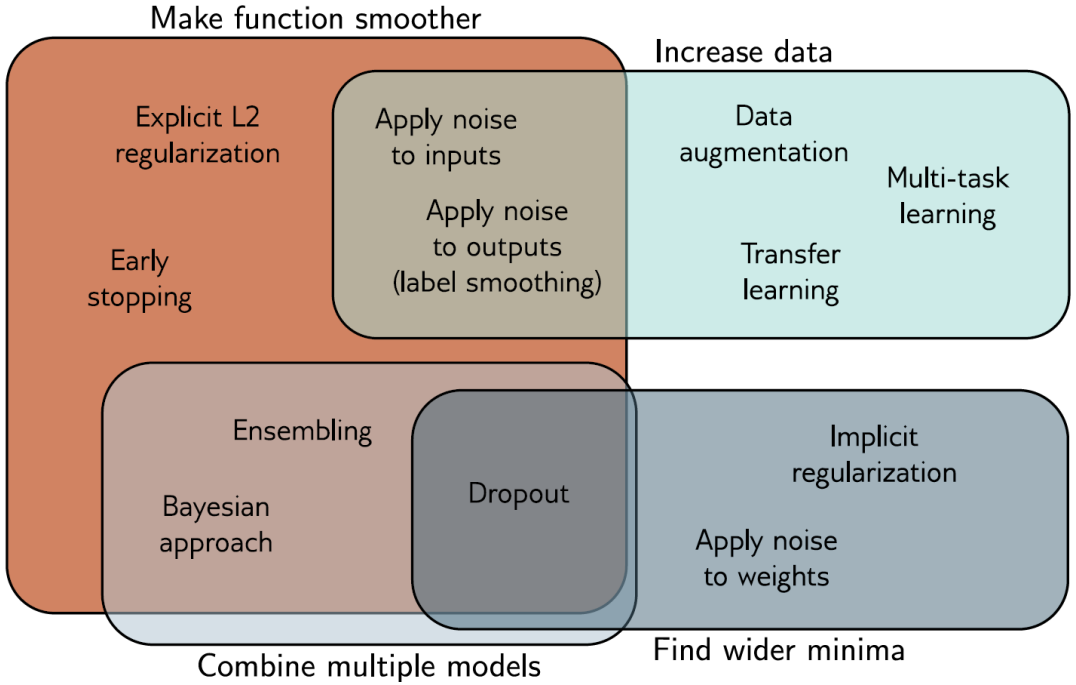
Supplement to General Recipe of Statistical Learning

adding one more piece to the general recipe of combining models, costs, and optimization methods: **regularization**

reduce test error, possibly at expense of increased training error

many ways:

explicit constraints, adding penalties, priors, parameter sharing, data set augmentation, early stopping, dropout, bagging, …



source

Idea of Regularization

3 possible scenarios:

trained model family …

1. excluded the true data-generating process (underfitting, high bias)
2. matched the true data-generating process
3. included the data-generating process, but also many others (overfitting, high variance)

goal of regularization: take a model from 3 into 2

L^2 Regularization (Ridge Regression)

adding **penalty term** on parameter L^2 norm to cost function

hyperparameter controlling strength of regularization

$$\tilde{J}(\hat{\boldsymbol{\theta}}) = J(\hat{\boldsymbol{\theta}}) + \lambda \cdot \hat{\boldsymbol{\theta}}^{\mathrm{T}} \hat{\boldsymbol{\theta}}$$

aka weight decay (neural networks)

corresponds to imposing constraint on parameters to lie in L^2 region (size controlled by λ)

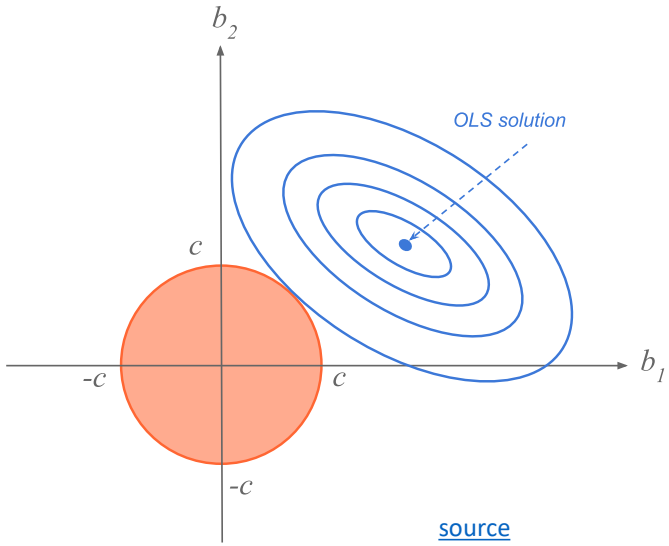
ridge regression for linear regression (MSE costs)

$$\tilde{J}(\hat{\boldsymbol{\beta}}) = (\boldsymbol{y} - \mathbf{X}\hat{\boldsymbol{\beta}})^{\mathrm{T}} (\boldsymbol{y} - \mathbf{X}\hat{\boldsymbol{\beta}}) + \lambda \cdot \hat{\boldsymbol{\beta}}^{\mathrm{T}} \hat{\boldsymbol{\beta}}$$

analytical solution:

$$\hat{\boldsymbol{\beta}}_{\text{ridge}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\, \mathbf{I})^{-1} \mathbf{X}^{\mathrm{T}} \boldsymbol{y}$$

$p \times p$ identity matrix

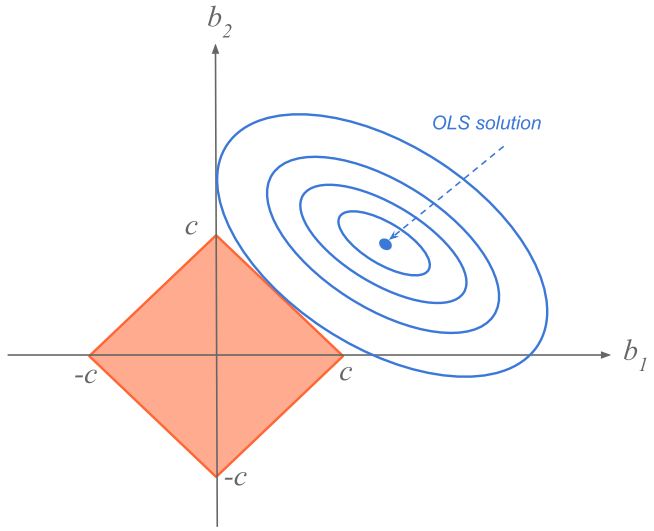


source

L^1 Regularization (LASSO)

adding **penalty term** on parameter L^1 norm to cost function

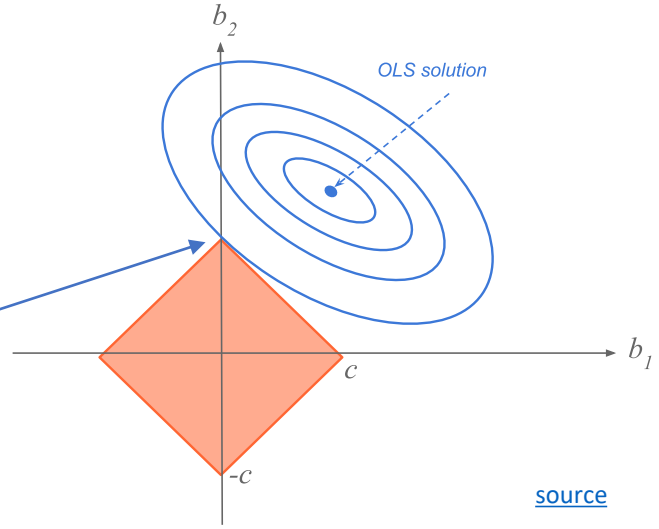
$$\tilde{J}(\hat{\boldsymbol{\theta}}) = J(\hat{\boldsymbol{\theta}}) + \lambda \cdot \sum_j |\hat{\theta}_j|$$



corresponds to imposing constraint on parameters to lie in L^1 region (size controlled by λ)

Least **A**bsolute **S**hrinkage and **S**election **O**perator:

also performs feature selection



source

Maximum a Posteriori Estimation

objective function

frequentist maximum likelihood estimation:

$$\widehat{\boldsymbol{\theta}}_{\mathrm{MLE}} = \mathrm{argmax}_{\widehat{\boldsymbol{\theta}}}\, \mathcal{L}\left(\widehat{\boldsymbol{\theta}}; \boldsymbol{y}|\boldsymbol{x}\right) = \mathrm{argmax}_{\widehat{\boldsymbol{\theta}}}\, p\left(\boldsymbol{y}|\boldsymbol{x}; \widehat{\boldsymbol{\theta}}\right)$$

maximum a posteriori estimation as alternative applying Bayes theorem:

$$\widehat{\boldsymbol{\theta}}_{\mathrm{MAP}} = \mathrm{argmax}_{\widehat{\boldsymbol{\theta}}}\, p\left(\widehat{\boldsymbol{\theta}}|\boldsymbol{y}, \boldsymbol{x}\right) = \mathrm{argmax}_{\widehat{\boldsymbol{\theta}}} \frac{p\left(\boldsymbol{y}|\boldsymbol{x}; \widehat{\boldsymbol{\theta}}\right) \cdot p\left(\widehat{\boldsymbol{\theta}}\right)}{\int p(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{\vartheta}) \cdot p(\boldsymbol{\vartheta})\, d\boldsymbol{\vartheta}}$$

$$= \mathrm{argmax}_{\widehat{\boldsymbol{\theta}}}\, p\left(\boldsymbol{y}|\boldsymbol{x}; \widehat{\boldsymbol{\theta}}\right) \cdot p\left(\widehat{\boldsymbol{\theta}}\right)$$

posterior distribution — likelihood — prior distribution

- approximation to Bayesian inference (using mode of posterior distribution)
- maximum likelihood special case of maximum a posteriori estimation with uniform prior

Priors as Regularizers

prior distribution: leveraging information that cannot be found in training data

many regularized estimation strategies can be interpreted as MAP

- L^2 regularization corresponds to MAP with Gaussian prior on parameters
- L^1 regularization corresponds to MAP with isotropic Laplace prior on parameters

usage of conjugate priors allows analytical solution of MAP (as mode of posterior distribution can be given in closed form)

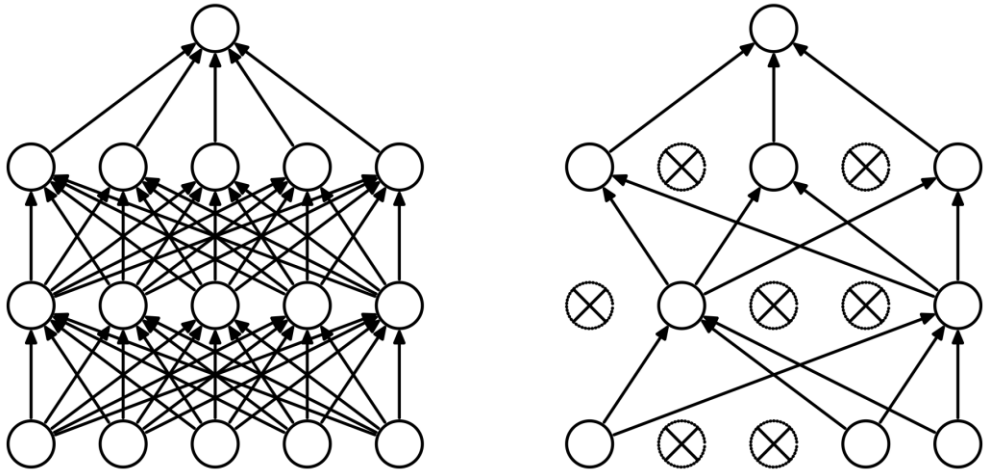
- conjugate distributions: prior $p(\boldsymbol{\theta})$ and posterior $p(\boldsymbol{\theta}|\boldsymbol{y}, \boldsymbol{x})$ in same probability distribution family (e.g., exponential family)
- $p(\boldsymbol{\theta})$ conjugate prior for likelihood $p(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{\theta})$

Dropout in Neural Networks

goal: prevent overfitting of large neural networks

idea: randomly drop non-output nodes (along with their connections) during training (not prediction)

- implicit ensemble method (kind of sampling from exponentially many differently thinned networks): for each mini-batch, randomly sample independent binary masks for the nodes
- inexpensive approximation to bagging (without training many different neural networks) with parameter sharing



(a) Standard Neural Net (b) After applying dropout.

source

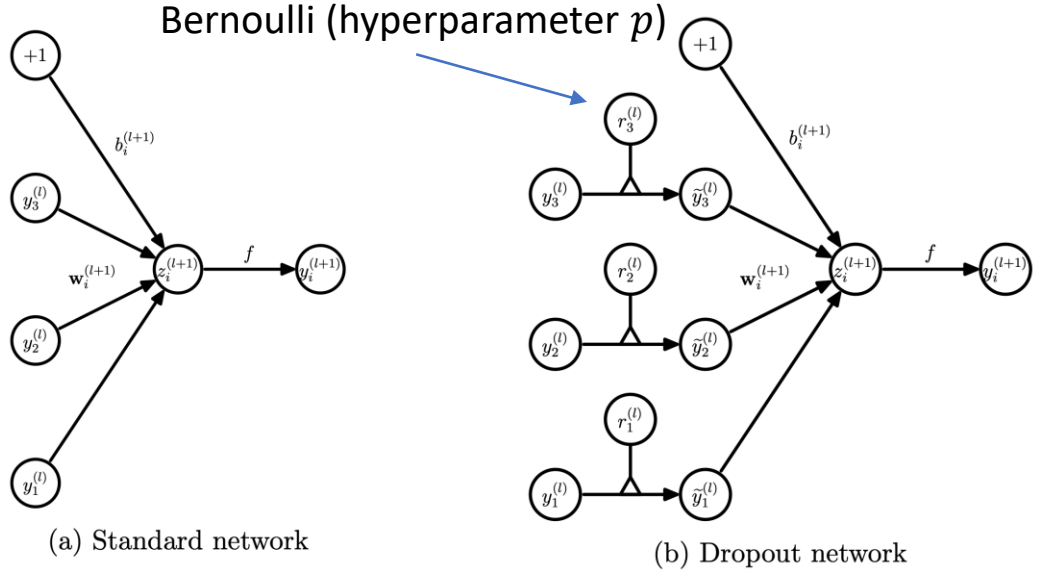
Dropout in Neural Networks

key advantage compared to averaging ensemble of independent models (typical bagging procedure):

better generalization by means of regularizing each hidden node to perform well regardless of which other hidden nodes are in the model (adaptability)

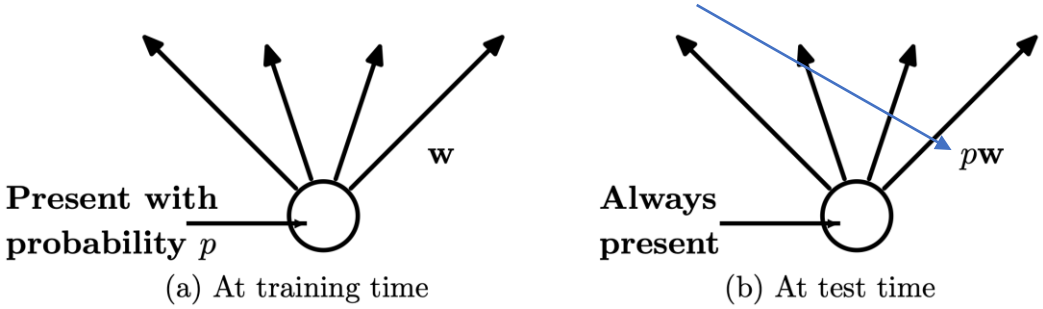
clever noise injection method: destroying extracted features rather than input values

Bernoulli (hyperparameter p)



(a) Standard network (b) Dropout network

weight scaling inference rule
(approximating ensemble predictions)



(a) At training time (b) At test time

source

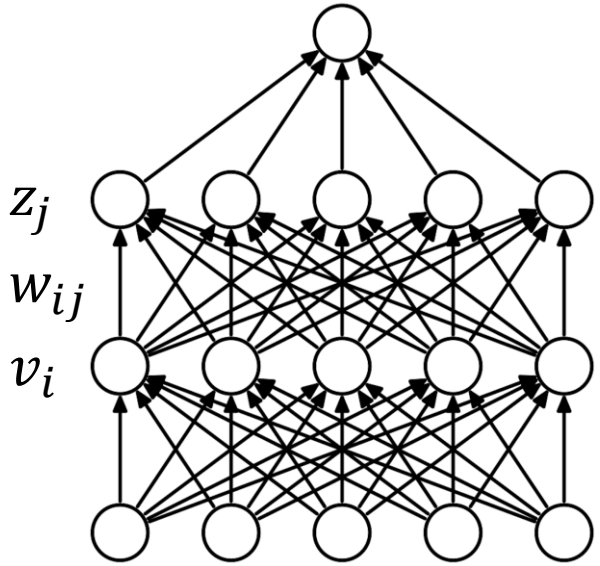
Convolutional Neural Networks (CNN)

Recap: Feed-Forward Neural Networks

computation in usual feed-forward network:

scalar input values z_j to activation function of nodes j in hidden or output layer as matrix multiplication of scalar output values v_i from activation function of nodes i from previous layer with connecting weights $w_{i,j}$

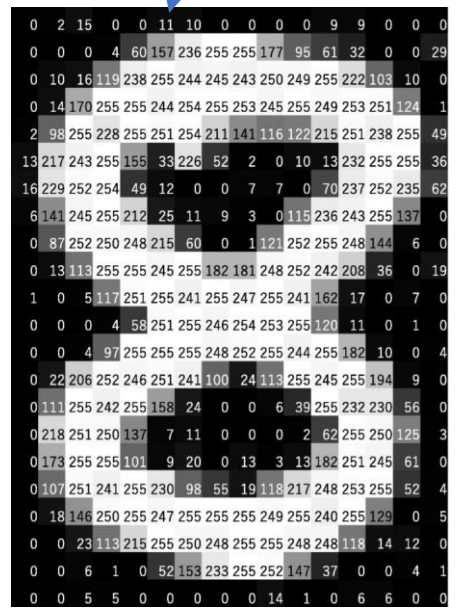
$$z_j = \sum_i v_i \, w_{i,j}$$



dropping dimension of different training observations in this view → loading full batch or mini-batches

Grid-Like Data

scalar value (like in usual feed-forward network)



source

data with grid-like topology (spatial structures), e.g.:

- time-series data: 1-D grid of data taken at regular time intervals (can also be done with recurrent neural networks or transformers)
- image data: 2-D grid of pixels (→ computer vision)

convolutional networks:

neural networks using convolutions with kernels (local groups of values, e.g., pixels from an object, highly correlated) in place of general matrix multiplications in at least one of their layers

→ highly regularized feed-forward networks

Convolution Operation

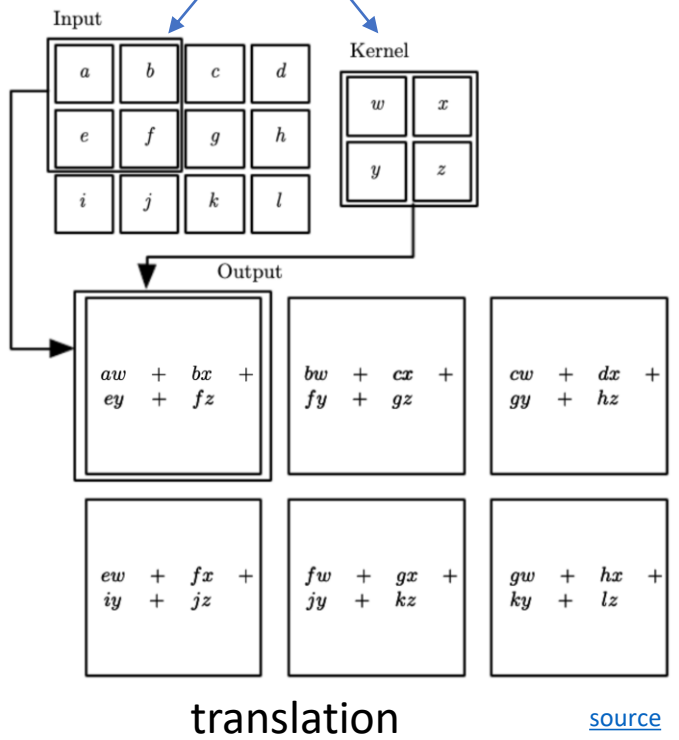
learned parameters:

- connection to **local patches** of previous layer's feature maps
- shared over entire image (common learning across image)

to be exact, usually rather cross-correlation instead of convolution operation (what would have – here):

$$Z_{i,j} = (K * V)_{i,j} = \sum_{m,n} V_{i+m,j+n}\, K_{m,n}$$

feature map — input (matrix) — kernel (matrix)



source

again, dropping dimension of different training observations

matrices → tensors: several input channels c (e.g., RGB) and several output channels f (different feature maps, e.g., vertical edge, nose, ear, …)

$$Z_{f,i,j} = \sum_{c,m,n} V_{c,i+m,j+n}\, K_{f,c,m,n}$$

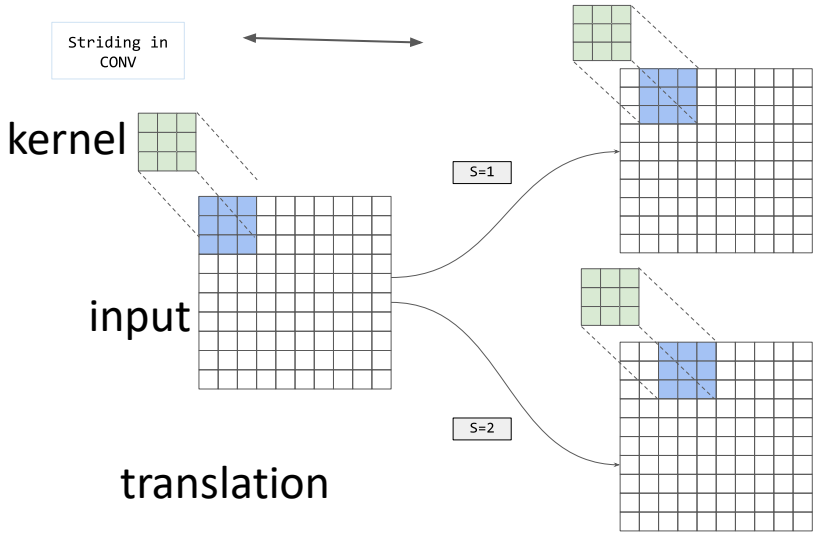
Important Details: Striding and Padding

need to define how to stride over image

$$Z_{f,i,j} = \sum_{c,m,n} V_{c,i\times s+m,j\times s+n} \, K_{f,c,m,n}$$

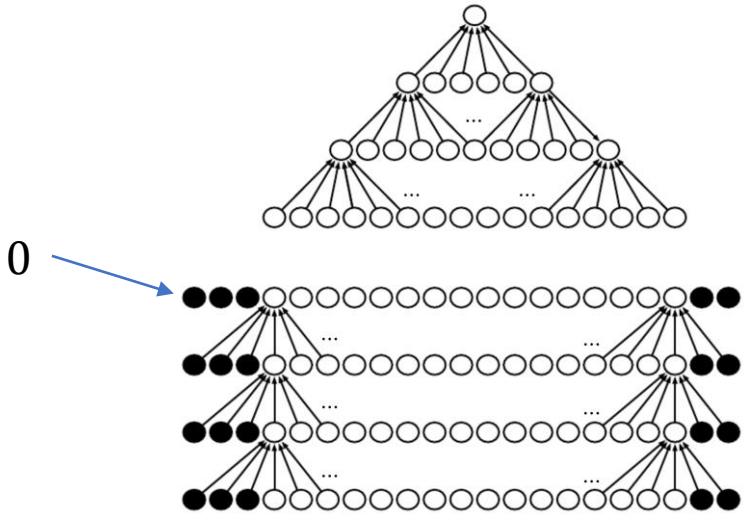
$s > 1$ corresponds to down-sampling

→ fewer nodes after convolutional layer



zero-padding of input to make it wider:

otherwise shrinking of represenation with each layer (depending on kernel size) → allowing large kernels and slow shrinkage

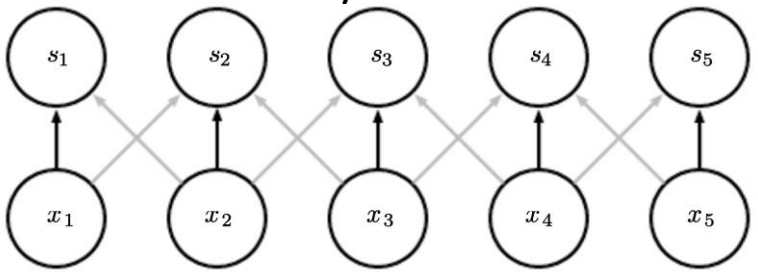


source

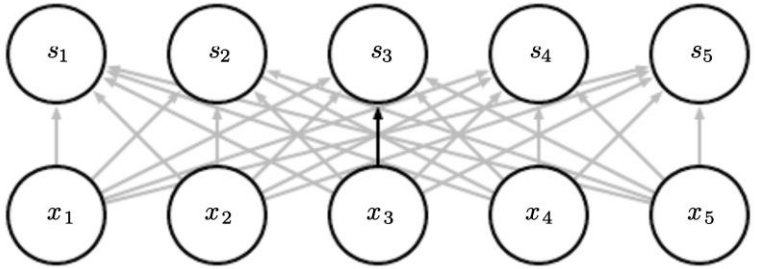
Regularization Effects

- sparse interactions: much less weights
- parameter sharing: use same weights for different connections

convolutional layer:



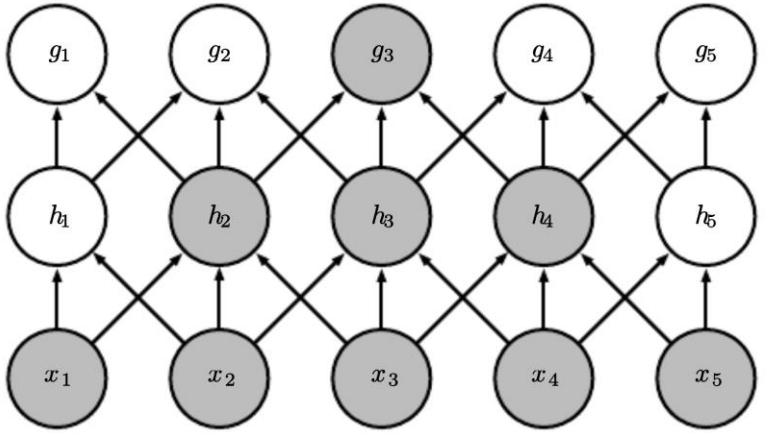
fully-connected layer:



effect of receptive field over several layers:

- consider only locally restricted number of input values from previous layer
- grows for earlier layers (indirect interactions)

→ hierarchical patterns from simple building blocks (many aspects of nature hierarchical)



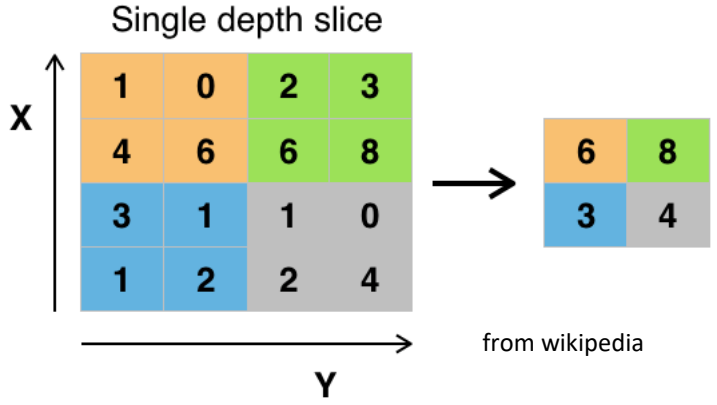
source

Another Ingredient: Pooling

replacing outputs of neighboring nodes with summary statistic (e.g., maximum or average value of nodes)

→ non-linear down-sampling (regularization)

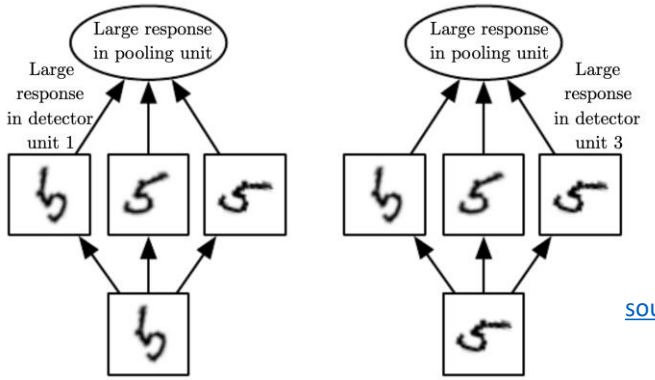
max pooling:



pooling is translation invariant: no interest in exact position of, e.g., maximum value

pooling over features learned by separate kernels (cross-channel pooling) can also learn other transformation invariances, like rotation or scale

(convolutions can detect same translated motif across entire image, but not rotated or scaled versions of it)

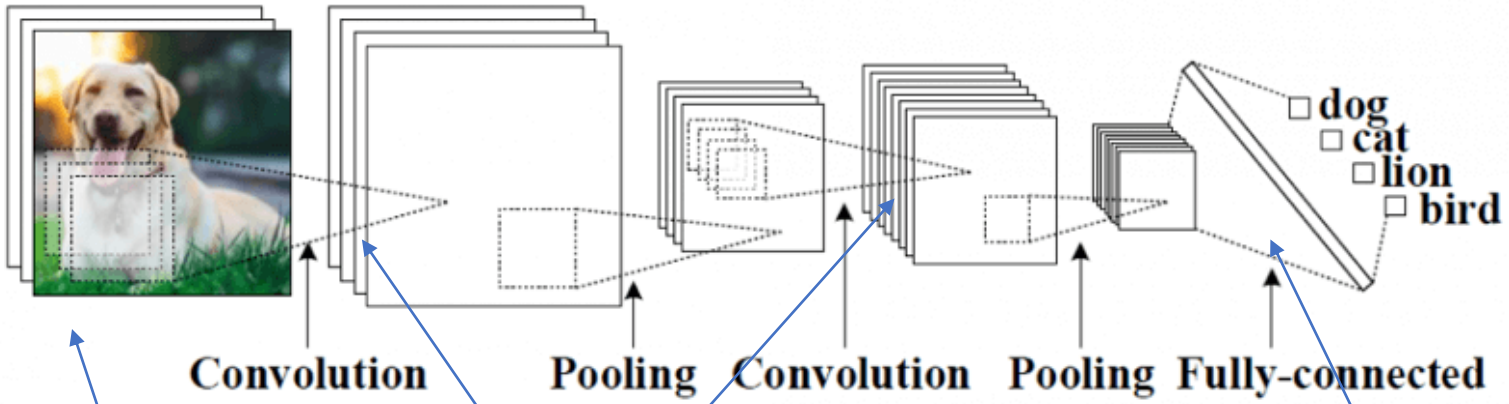


source

Putting It All Together

CNN in short:
local connections, shared weights, pooling, many layers

kernels

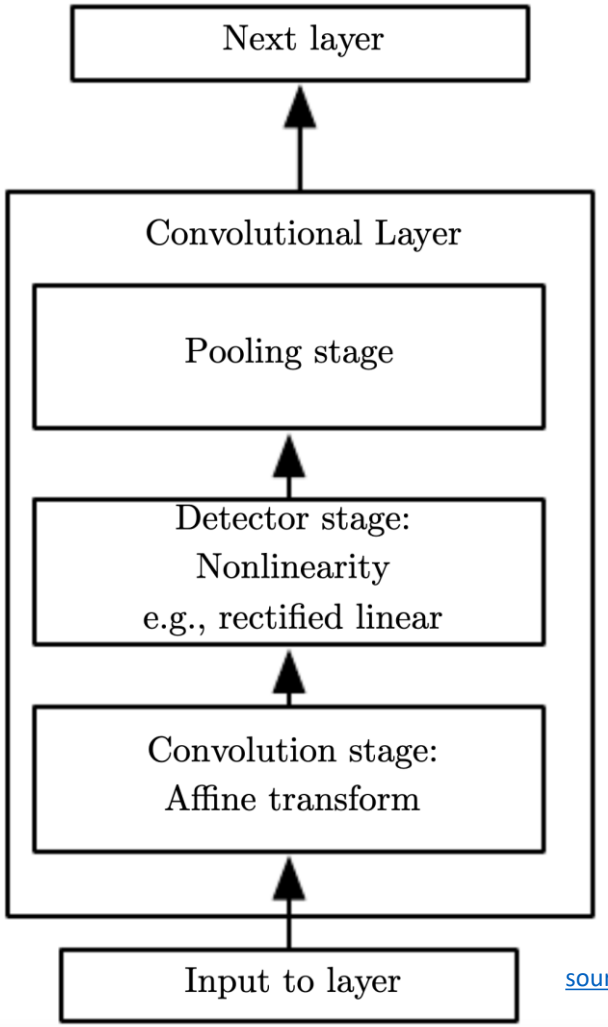


many images (training examples), potentially with several channels

several kernels producing several feature maps

down-sampling by convolutions and pooling

flatten dimensions for final classification or regression



source

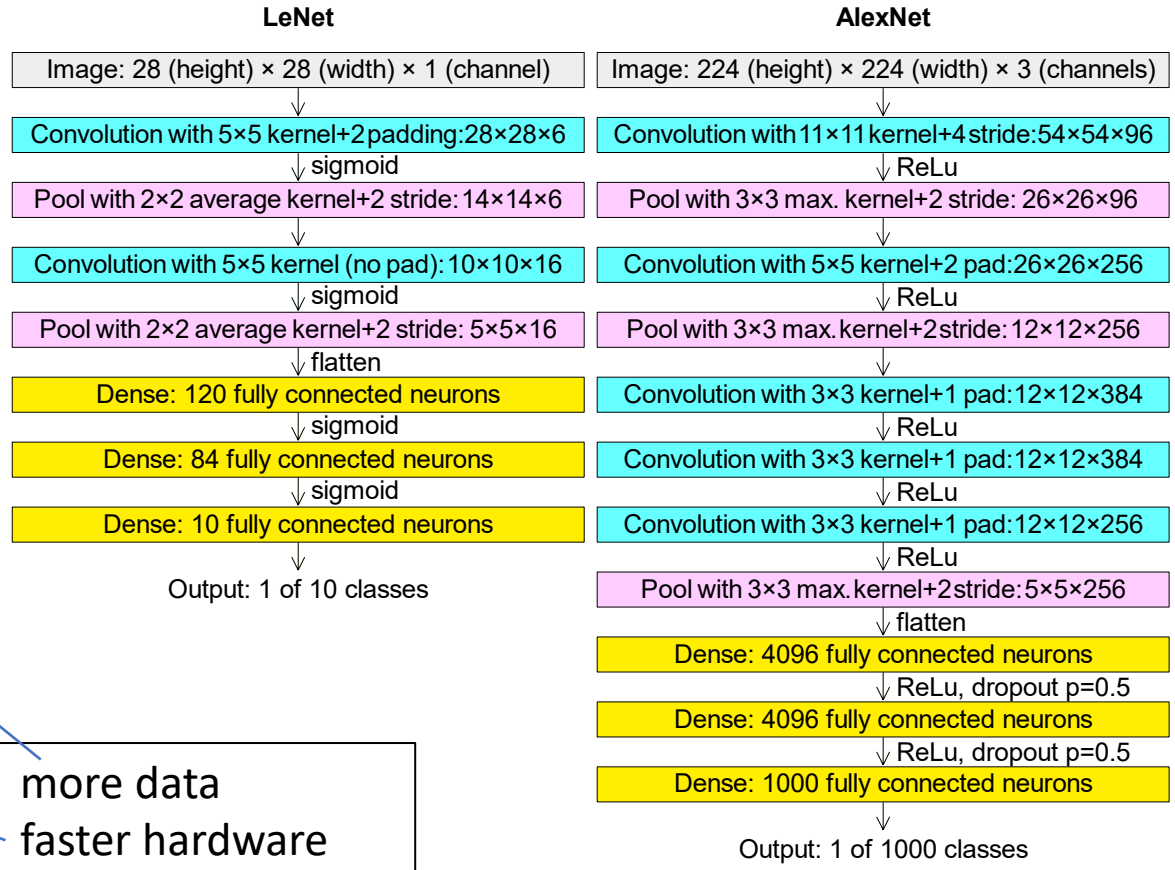
The Pivotal Moment: AlexNet

AlexNet finally started deep learning hype (won ImageNet challenge 2012)

some major improvements:

- GPU implementation (enabling more layers → better hierarchical representation)
- using ReLU activation functions
- using dropout

- more data
- faster hardware
- smarter algorithms



from wikipedia

Translation Symmetries

invariance: same output regardless how input is shifted

in CNN: pooling

example: object detected, no matter if in upper left or lower right corner

equivariance: shift of input corresponds to shift of output

in CNN: convolution

example: feature maps conserving spatial information of inputs (e.g., relative positions of eyes and nose useful for later face recognition)

convolution and pooling can be interpreted as infinitely strong priors

- locality
- convolution equivariant to translation
- pooling invariant to translation

} spatial inductive bias

Inductive Bias

Inductive Bias (aka Learning Bias)

set of assumptions that a learning algorithm uses to predict outputs of inputs that it has not encountered during training

examples: linear response (linear regression), maximum margin (SVM), nearest neighbors (kNN), spatial structure (CNN)

but also: different regularization and optimization methods

crucial piece of generalization

data in disguise (replacement for missing information on specific situations in limited training data sets)

No Free Lunch Theorem

All optimization/ML algorithms (both sophisticated and simple ones) perform equally well when their performance is averaged across all possible problems.

(But deep learning is trying to solve many problems with very general-purpose forms of regularization.)

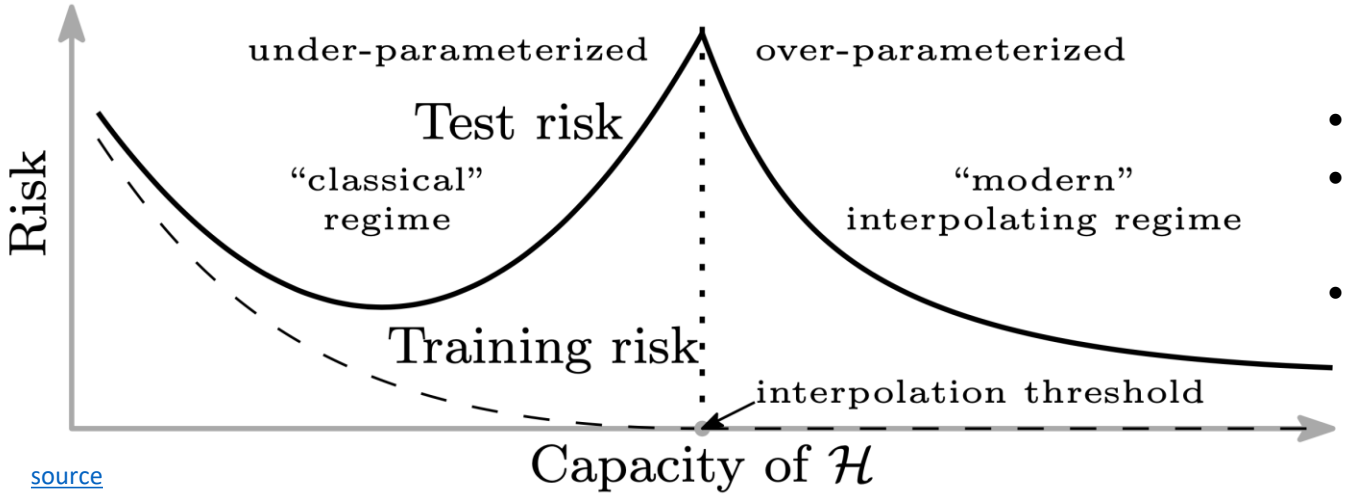
Particularly, model complexity doesn't reflect if inductive bias is appropriate for problem at hand.

→ choose right ML method for learning task at hand

https://arxiv.org/abs/1812.11118
https://arxiv.org/abs/1912.02292
https://arxiv.org/abs/2201.02177

Double Descent in Deep Learning

deep neural networks show additional structure for many parameters:



source

- generalization error first decreases
- then peaks (sharp increase in variance) at the interpolation threshold (where training error vanishes)
- then decreases monotonically again

- peak can be suppressed by regularization
- but potentially also suppression of second descent in case of sub-optimal inductive bias

(supposedly) not only Deep Learning: sufficiently complex, i.e., over-parametrized, ML models

Interpretation of Double Descent

all models right of interpolation threshold fit training data perfectly

find smoothest function that perfectly fits observed data (best inductive bias for task at hand)

→ a form of Occam's razor (the simplest explanation compatible with the observations should be preferred)

larger function classes contain more interpolating functions for training data at hand, i.e., better chance to include simpler/smoother ones (smaller norm)

→ increasing model complexity can improve generalization (but need more data)

recipe: over-parametrize and regularize

Memorization First Step Toward Generalization?

many interpolating models (perfectly fitting training data) in over-parameterization regime

empirical evidence that explicit regularization is not necessary for generalization (but helps to choose smooth solution)

→ implicit regularization (tendency of an algorithm to seek out solutions that generalize well on its own)

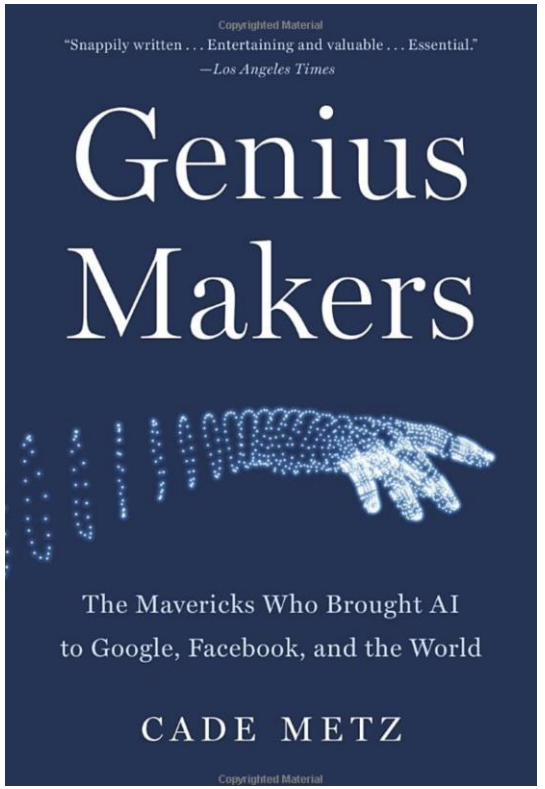
running **Stochastic Gradient Descent** to convergence: pick minimum norm solution of different interpolating models → implicitly regularizing

Literature

papers:

- CNN: neural networks work (at least for LeCun ☺)
- AlexNet: deep learning takes over
- dropout

nice historical overview:



Key Challenge for AI: Generalization

common sense of humans or animals enabled by world model (generalized representation of the world, allows to be inattentive to vast amount of irrelevant details)

not yet reached this level of generalization in AI systems

without common sense: need for tons of data and hard-wired engineering of corner cases (example: self-driving cars)