Other Nonlinear Supervised Learning Methods

Johanni Brea

Introduction to Machine Learning

Inductive Bias of Different Nonlinear Methods

With sufficiently large neural networks one can approximate any function, and, hence, solve any regression/classification problem.

However, hyper-parameter tuning can be tricky and other non-linear methods may be faster in finding a good fit for some datasets.

Why?

Each machine learning method has an **inductive bias**.

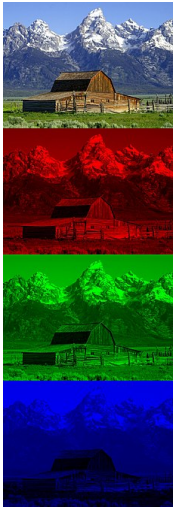
The **inductive bias** (or learning bias) of a learning algorithm is the set of assumptions that the learner uses to predict outputs of given inputs that it has not encountered.

For any given dataset, the inductive bias of method A can be better than the inductive bias of method B.

There is not a single method that works best for every dataset.

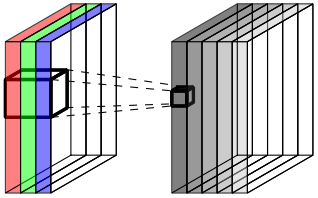
Table of Contents

Convolution Volumes



- Color images can be represented with 3 color channels, i.e. a 200×200 color images can be represented as $200 \times 200 \times 3$ real values.
- Instead of flattening the 3 color channels into one long vector, arrange them in a 3D volume (grayscale images: 2D volume)
- "convolutional neurons" apply the same local operation to all x-y positions.

From 3 (colour) channels to 5 (filter) channels



Convolutional Layer

A convolutional layer l with $K^{(l)}$ filters of size $(2f+1, 2f+1, K^{(l-1)})$computes

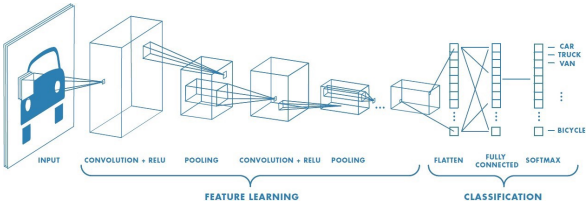
$$a^{(l)}_{xy,k} = g^{(l)}\left(w^{(l)}_{k0} + \sum_{q=-f}^{f}\sum_{r=-f}^{f}\sum_{s=1}^{K^{(l-1)}} w^{(l)}_{qrs,k} a^{(l-1)}_{x+q,y+r,s}\right),$$

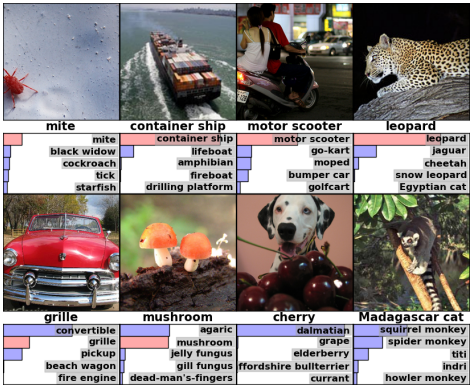
where $k = 1, \ldots, K^{(l)}$, $a^{(l-1)}_{x+q,y+r,s}$ are the activities in the previous layer, $w^{(l)}_{k0}$ is the bias of filter k, $w^{(l)}_{qrs,k}$ are the weights of filter k.

Convolutional Networks

winner of imagenet competition 2012

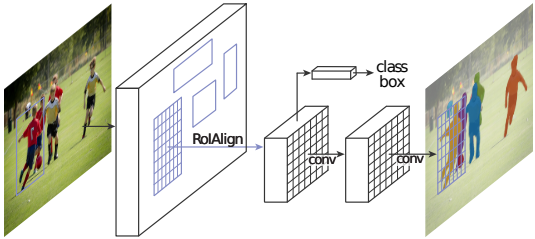
input: colour images $224 \times 224 \times 3$
output: 1000 classes
hidden layers: 5 convolutional layers,
3 dense layers





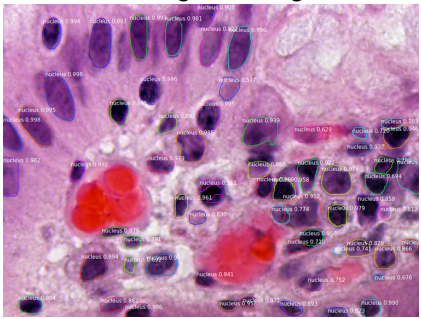
ImageNet Classification with Deep Convolutional Neural Networks, Krizhevsky, Sutskever, Hinton, 2012

Convolutional Networks: Mask R-CNN



$$\mathcal{L}^{\text{total}} = \mathcal{L}^{\text{class}} + \mathcal{L}^{\text{box}} + \mathcal{L}^{\text{mask}}$$

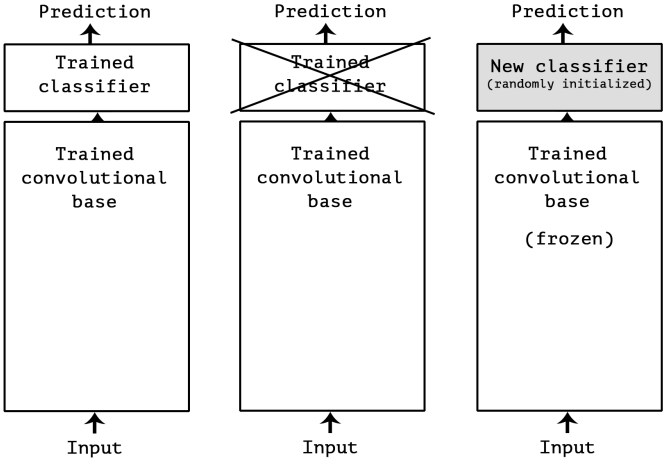
Nuclei Counting and Segmentation



Mask R-CNN, He Gkioxari, Dollár, Girshick, 2018
`https://github.com/matterport/Mask_RCNN`

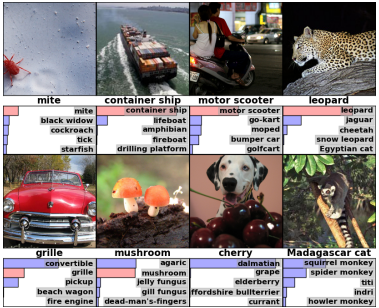
Table of Contents

Transfer Learning: Reusing Learned Features



Transfer Learning: Reusing Learned Features

Training Set for Features
Imagenet Dataset: more than 1 million images 1000 categories



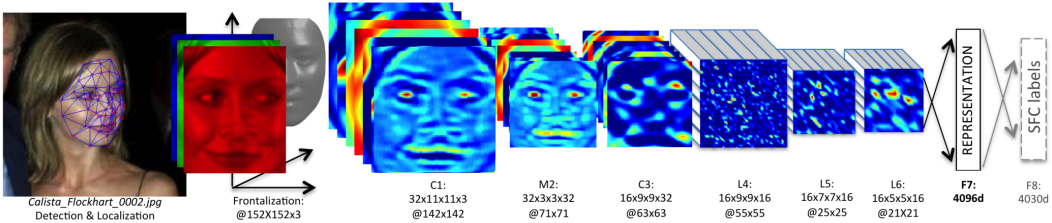
Training Set for Classifier
102 Category Flower Dataset (≈ 10'000 images)

102 Category Flower Dataset

Category	#ims	Category	#ims	Category	#ims
 alpine sea holly	43	 buttercup	71	 fire lily	40
 anthurium	105	 californian poppy	102	 foxglove	162
 artichoke	78	 camellia	91	 frangipani	166
 azalea	96	 canna lily	82	 fritillary	91

Transfer Learning: Reusing Learned Features

DeepFace: Closing the Gap to Human-Level Performance in Face Verification (2014)



- 4 million user-labeled faces on FaceBook images of 4000 individuals
- Retrain fully-connected layers at the top on Labeled Faces in the Wild (LFW) dataset reaching (human level) accuracy of 97.35%

Table of Contents

Recurrent Neural Networks(RNN)

Let us assume time dependent inputs $x(t) = (x(0), x(1), \ldots, x(T))$.
In a recurrent neural network the activity $a^{(l)}(t)$ of the hidden neurons at time t does not only depend on the input $x(t)$ at t, but also on the previous hidden activity $a^{(l)}(t-1)$, e.g.

$$a^{(l)}(t) = g^{(l)}(w^{(l)}a^{(l-1)}(t) + w^{(l,\text{rec})}a^{(l)}(t-1) + b^{(l)})$$

See also the excellent blog post
`https://karpathy.github.io/2015/05/21/rnn-effectiveness/`

Application of a Recurrent Neural Network for Language Detection

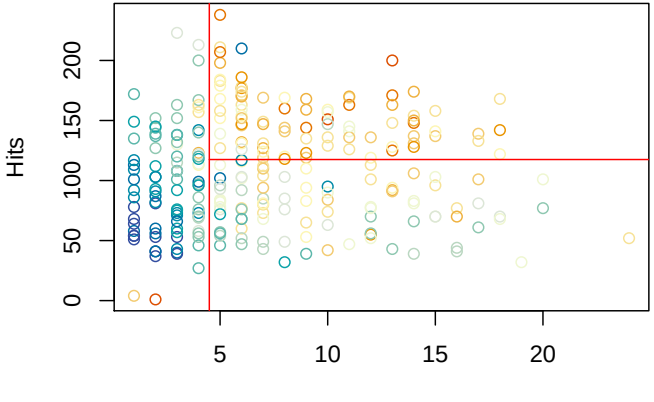
inputs: sentences, output: language label

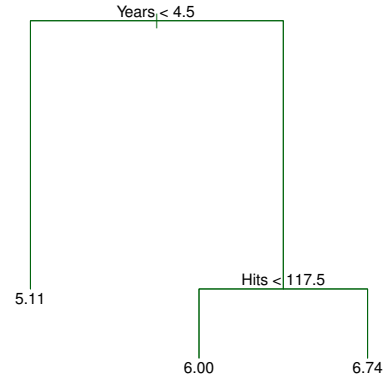
time step	1	2	3	4	5	6	7	8	9	10	11	12
character	t	o		b	e		o	r		n	o	t
	0	1	0	0	0	0	1	0	0	0	1	0
1-hot input	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	1	0	0	0	0	0	0	0	0	0	0	1
	0.1	0.4	0.0	0.8	0.7	0.1	0.6	0.0	0.0	0.0	0.0	0.0
hidden layer	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	0.3	0.7	1.0	0.3	0.0	0.8	0.3	0.3	1.1	1.3	1.3	1.4

Final hidden activity depends on whole phrase; can be used to predict the language.
All weights are jointly trained with gradient descent to maximize the likelihood function.

Table of Contents

Example: Baseball Player Salary Prediction with Regression Trees

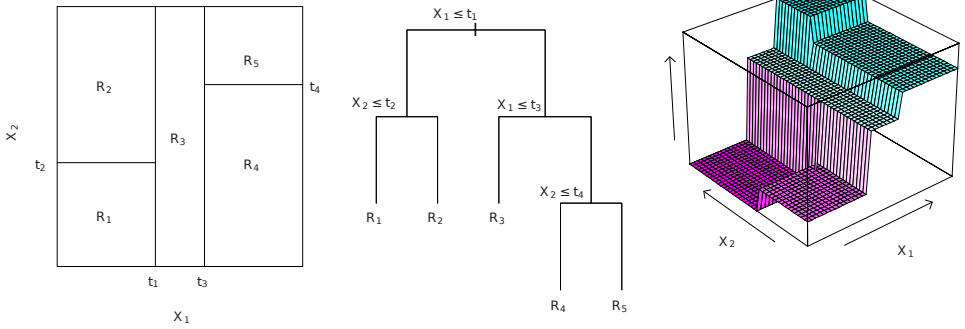




Leaf values: `log(Salary)`

Prediction: Players with $>$ 4.5 years of experience and $>$ 117.5 hits have an average salary of $\exp(6.74) \approx 845$ thousand USD.

Regression Trees



Tree with 4 **internal nodes** and 5 **terminal nodes** or **leafs**.

Recursive Binary Splitting

1. Start with the root of the tree, i.e. the full predictor space.
2. Define $R_1(j, s) = \{X | X_j < s\}$ and $R_2(j, s) = \{X | X_j \geq s\}$ and seek j and s that minimize

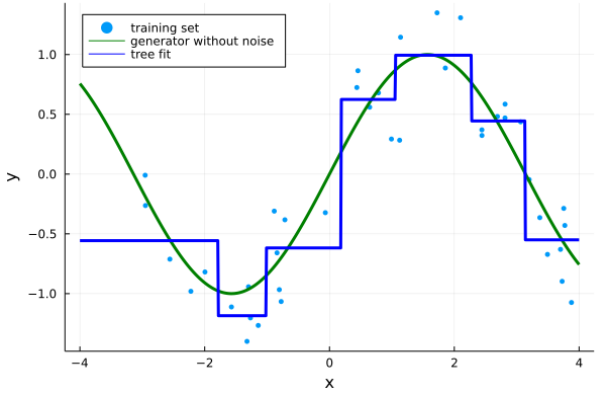
$$\sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

3. Repeat the splitting process on each of the two regions.
4. Iteratively split resulting regions until some stopping criterion is met; e.g. until no region contains more than 5 observations.

Note: Other loss functions $L(y, \hat{y})$ can be used instead of the squared error, e.g. negative log-likelihood loss for binary/multinomial/Poisson regression.

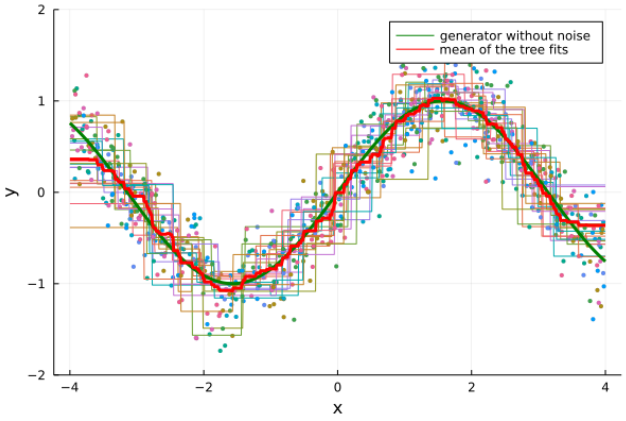
Recursive Binary Splitting: Example

Example: Fitting a Decision Tree



In one input dimension the result of a decision tree regression looks like the result of a zeroth order spline, where the knots are found automatically.

Ensembling



Averaging individual fits with low bias but high variance can lead to an improved prediction.

See also `https://www.kaggle.com/code/amrmahmoud123/1-guide-to-ensembling-methods`

Random Forests

$$f(x) = \frac{1}{K}\sum_{k=1}^{K} \text{tree}_k(x)$$

Each tree $\text{tree}_k(x)$ is fitted to a separate bootstrap sample of the data; for each split, only a fraction of all features are candidates for a split.

Important Hyperparameters:

- Maximal depth of the trees.
- Number of trees.
- Sampling fraction of the data.
- Number of features to select at random.

1. Input: data $\mathcal{D} = ((x_1, y_1), \ldots, (x_n, y_n))$, loss function L, number of trees K, number of features m (e.g. $m = \sqrt{p}$), tree growth stopping criterion, (sampling fraction of data, unless bootstrapping is used)
2. For $k = 1, \ldots K$
   - obtain $\mathcal{D}_k$ by bootstrapping or sampling a fraction of the data.
   - grow tree_k on $\mathcal{D}_k$: choose at every split m predictors uniformly random as candidates for splitting.
3. Return all trees.

Gradient Boosting Trees

Gradient Boosting Trees (GBT):

$$f(x) = \sum_{k=1}^{K} a_k \texttt{tree}_k(x)$$

The trees $\texttt{tree}_k$ and the a_ks are learned sequentially. Idea: $\texttt{tree}_i$ should boost the performance by compensating for the mistakes made by $\sum_{k=1}^{i-1} a_k \texttt{tree}_k(x)$.

Important Hyperparameters:

- Maximal depth of the trees.
- Number of trees (also called "number of rounds").
- Learning rate ("eta")

Example: Boosting for Regression Trees

In a regression setting with RSS loss $L = \sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2$

1. Start with a weak expert $f_0(x) = \frac{1}{n}\sum_{i=1}^{n} y_i$
2. For $m = 1$ to B
   a) For $i = 1, 2, \ldots, n$ compute residuals $r_{im} = y_i - f_{m-1}(x_i)$
   b) Fit a regression tree to r_{im}. Call this tree $\Delta_m(x)$.
   c) Update $f_m(x) = f_{m-1}(x) + \lambda\Delta_m(x)$
3. Return $\hat{f}(x) = f_B(x)$.

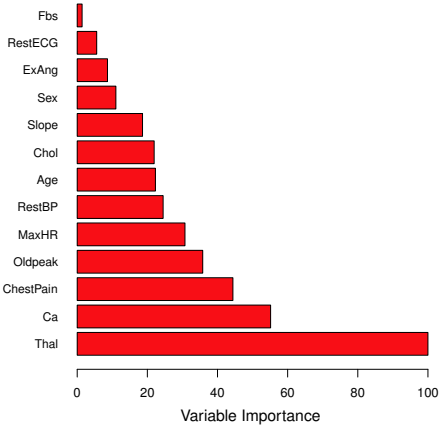
Why does it work?

Intuition: if $\Delta_m(x)$ in 2b) perfectly fits the residuals and $\lambda = 1$ we have

$$f_m(x_i) = f_{m-1}(x_i) + \Delta_m(x_i) = f_{m-1}(x_i) + r_{im} = f_{m-1}(x_i) + y_i - f_{m-1}(x_i) = y_i$$

With non-perfect fits and $\lambda < 1$ the result moves closer to y_i in each iteration.

Variable Importance Plots



For Random Forests or Gradient Boosting

- The variable importance indicates how much the loss decreased in all splits along a given predictor dimension, considering all trees.
- It is usually expressed relative to the maximum.
- In the example on the left, splits along the Thal direction led to the largest decrease in loss.

Some of the figures in this presentation are taken from "An Introduction to Statistical Learning, with applications in R" (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani

Summary

- Tree-based methods split the input space parallel to the axes.
  This is similar to spline-based methods or neural networks with the heaviside activation function and vector features parallel to the axes (only one of the weight entries is non-zero). General neural networks are more flexible.
- The Bootstrap is a method to obtain multiple training sets from a single training set. Averaging models over bootstrapped training sets (bagging) can improve performance.
- Ensembling: flexible, individual models with high variance and low bias can be averaged to obtain good fits.
  Generalization: One can also (weighted) average models of different kind, e.g. neural networks with random forests.

Table of Contents

Support Vector Machines

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \kappa(x_i, x)$$

- $\kappa(x, y)$ is a called kernel; we can choose the type of kernel, e.g. radial kernel $\kappa(x, y) = \exp(-\gamma \sum_{i=1}^{p} (x_i - y_i)^2)$.
- β_0 and α_i are learned with a special loss function (not maximum likelihood).
- Most α_i's are zero after training.
- The training points x_i with non-zero α_i are called support vectors.

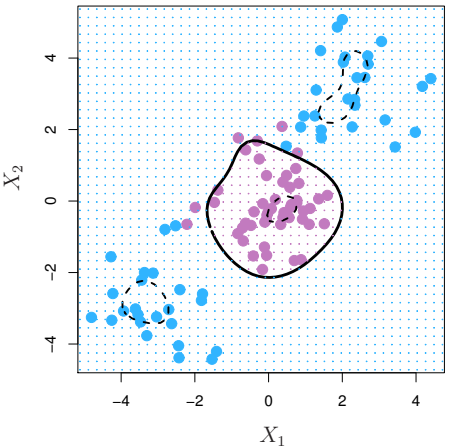
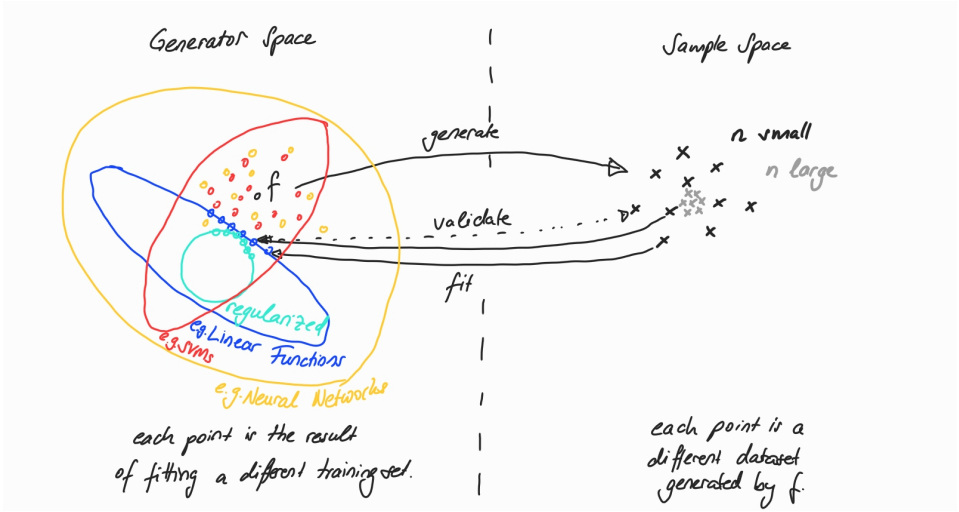


Table of Contents

Supervised Learning: the Big Picture



Supervised Learning: the Big Picture

Lookup Table	Function Approximation
Only useful when the training set contains all inputs that will ever appear	Allows for generalization to unseen input. How good the generalization is depends on the data and the inductive bias, i.e. generalization is good when the function approximator comes close to the true data generator.

Non-Parametric Function Approximation	Parametric Function Approximation
The training set is explicit stored and used to make predictions, e.g. KNN; typically fast at training, slow at prediction	The training set is implicitly stored in the parameters, e.g. linear regression, neural networks, trees; typically slow at training, fast at prediction

Additional Considerations:

- What is a good noise model for the conditional $p(Y|X)$? E.g. normal, log-normal, Poisson, ...
- Is there known structure in the data that can be exploited?

Supervised Learning: the Big Picture

In general it is not clear which method should be chosen for which task and data set.

Rules of Thumb

- Images as input: convolutional neural networks or transformers
- Sequences as input (text, music): transformers (or recurrent neural networks)
- With Gradient Boosting Trees or Random Forests one can often get decent results in little time (not much tuning is required). But well tuned neural networks tend often to be better.
- Carefully tuned regularization (L1, L2, early stopping or dropout) usually matters a lot.

Suggested Reading

- 10.3. Convolutional Neural Networks
- 8.1. The Basics of Decision Trees
- 8.2. Bagging, Random Forests, Boosting (without 8.2.4)
- (optional) 10.5. Recurrent Neural Networks
- (optional) 9 Support Vector Machines