Basic introduction to machine learning

Jeffrey A. Fessler

EECS Department, BME Department, Dept. of Radiology
University of Michigan

ISMRM Educational Course:
Machine Learning: Everything You Wanted to Know but Were Afraid to Ask

2021-05-15

Declaration: No relevant financial interests or relationships to disclose

Outline

Outline

Slides and code

J. Fessler

https://tinyurl.com/ml2-18-jf

- Slides with bibliography
- Jupyter notebook
  - Julia code for all figures shown
  - Ju=Julia py=python r=R
  - Julia 1.0 released Aug. 2018
  - SIAM Review paper [1]
  - Convenience of scripting, performance of compiled code

Machine learning definitions

- `https://en.wikipedia.org/wiki/Machine_learning` 2021-04-16:

  Machine learning (ML) is the study of computer algorithms that improve automatically through experience and by the use of data.

Machine learning definitions

- https://en.wikipedia.org/wiki/Machine_learning 2021-04-16:

  Machine learning (ML) is the study of computer algorithms that improve automatically through experience and by the use of data. ML algorithms build a model based on sample data, known as "training data," to make predictions or decisions without being explicitly programmed to do so.

Machine learning definitions

- `https://en.wikipedia.org/wiki/Machine_learning` 2021-04-16:

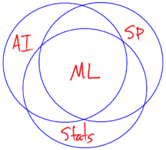
  Machine learning (ML) is the study of computer algorithms that improve automatically through experience and by the use of data. ML algorithms build a model based on sample data, known as "training data," to make predictions or decisions without being explicitly programmed to do so.

Machine learning definitions

- https://en.wikipedia.org/wiki/Machine_learning 2021-04-16:

  Machine learning (ML) is the study of computer algorithms that improve automatically through experience and by the use of data. ML algorithms build a model based on sample data, known as "training data," to make predictions or decisions without being explicitly programmed to do so.

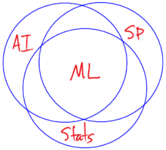
- Statistical perspective: "Machine learning is a field of study concerned with making quantitative inferences and predictions based on data." (Clay Scott, 2016)



- https://en.wikipedia.org/wiki/Machine_learning 2021-04-16:

  Machine learning (ML) is the study of computer algorithms that improve automatically through experience and by the use of data. ML algorithms build a model based on sample data, known as "training data," to make predictions or decisions without being explicitly programmed to do so.

- Statistical perspective: "Machine learning is a field of study concerned with making quantitative inferences and predictions based on data." (Clay Scott, 2016)

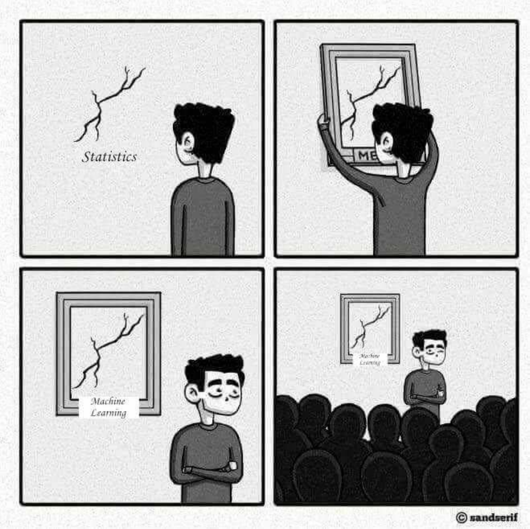


- ML is statistics without confidence intervals, p-values, or control of Type-I/II errors?

ML definitions

Image credit:

https://www.reddit.com/r/ProgrammerHumor/comments/88o6an/machine_learning/



Machine learning categories

Application:

- classification (labeling / detection / segmentation)
- regression (parameter estimation / quantification)

Machine learning categories

Application:

- classification (labeling / detection / segmentation)
- regression (parameter estimation / quantification)

Training method:

- supervised learning (labeled training data)
- unsupervised learning
- semi-supervised learning
- reinforcement learning

ML categories

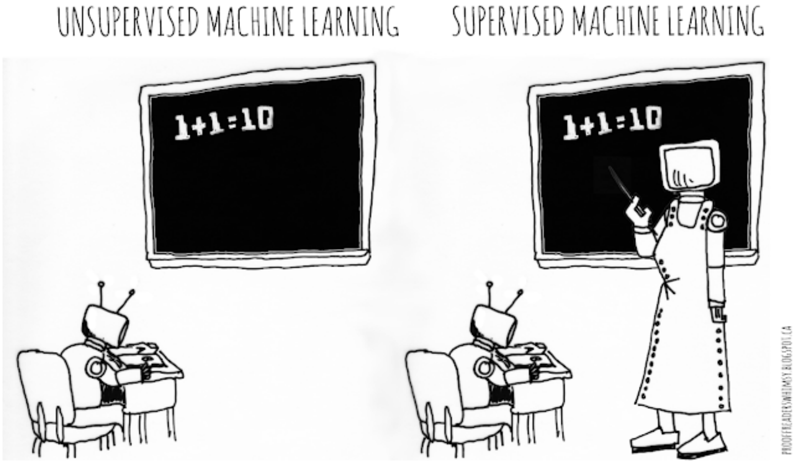
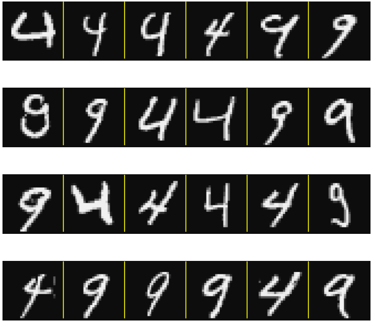


Image credit: `http://prooffreaderswhimsy.blogspot.com/2014/11/machine-learning.html`

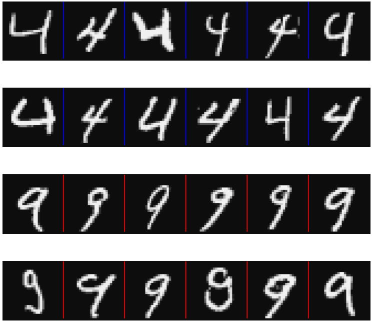
Unsupervised vs Supervised Learning

J. Fessler

Unsupervised



Supervised



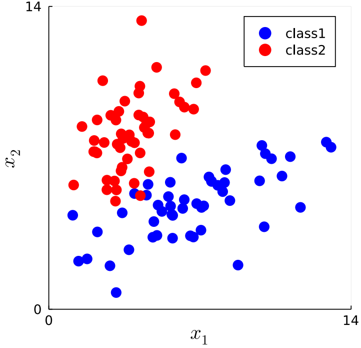
Domain experts needed...

Supervised learning overview: classification

Given paired (feature,label) training data: $(\boldsymbol{x}_1, y_1), \ldots (\boldsymbol{x}_N, y_N)$

Example:

- $\boldsymbol{x} \in \mathbb{R}^2$
- $y \in \{\text{class1=blue}, \text{class2=red}\}$

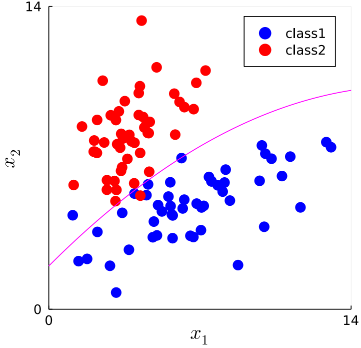


Supervised learning overview: classification

Given paired (feature,label) training data:
$(\boldsymbol{x}_1, y_1), \dots (\boldsymbol{x}_N, y_N)$

Goal: predict output (*e.g.*, class) y
for a subsequent test feature $\boldsymbol{x}$

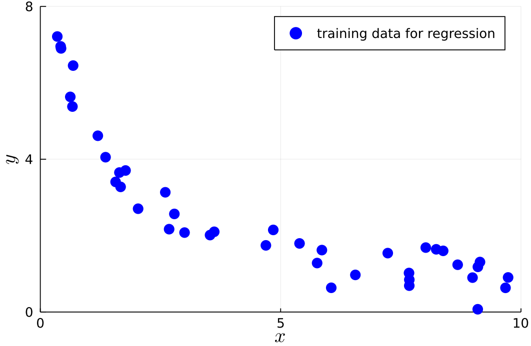
A classifier is a function $y = f(\boldsymbol{x})$ that maps
a feature vector into a class label,
i.e., $f : \mathbb{R}^d \mapsto \{1, \dots, K\}$.



Given paired (feature,label) training data: $(\boldsymbol{x}_1, y_1), \ldots (\boldsymbol{x}_N, y_N)$.

Example:

- $\boldsymbol{x} \in \mathbb{R}$
- $y \in \mathbb{R}$

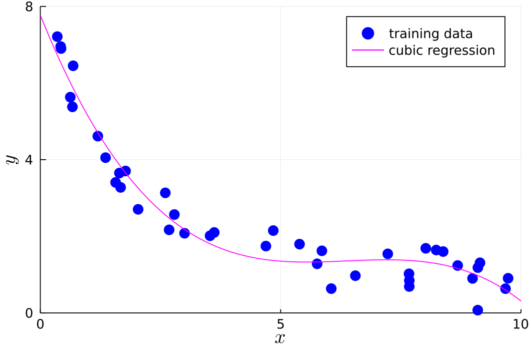


Supervised learning overview: regression

Given paired (feature,label) training data: $(\boldsymbol{x}_1, y_1), \ldots (\boldsymbol{x}_N, y_N)$.

Goal: predict output (*e.g.*, value) y for a subsequent test feature $\boldsymbol{x}$.

Key challenge in supervised learning is generalization beyond training data for future predictions.

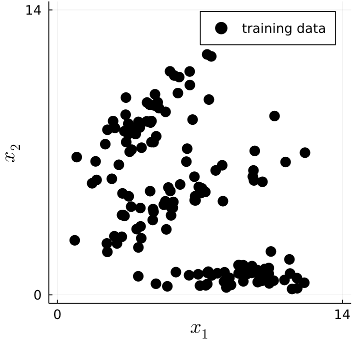


Unsupervised learning overview: clustering

No labels, just feature vector training data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$.

Example:

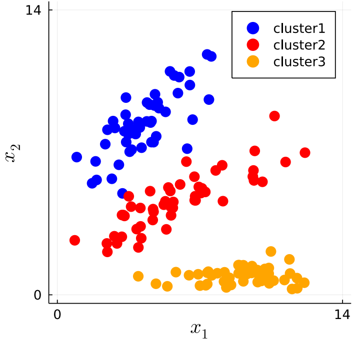
- $\boldsymbol{x} \in \mathbb{R}^2$



No labels, just feature vector training data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$.

Goal: understand data structure

- Clustering
- Dimensionality reduction
- Density estimation

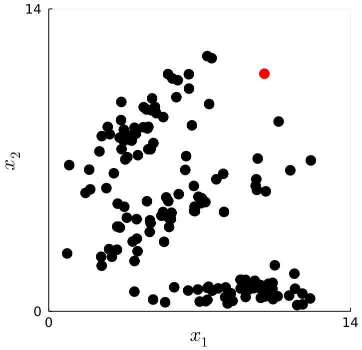


Unsupervised learning overview: clustering

No labels, just feature vector training data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$.

Another unsupervised learning problem: novelty detection.

Many other ML problems...



More categories of ML methods

Distribution assumptions

- Generative: full probabilistic model for data
- Discriminative: partial or no probabilistic model

Model type / complexity:

- parametric: number of model parameters is independent of sample size
- nonparametric: number of model parameters grows with sample size

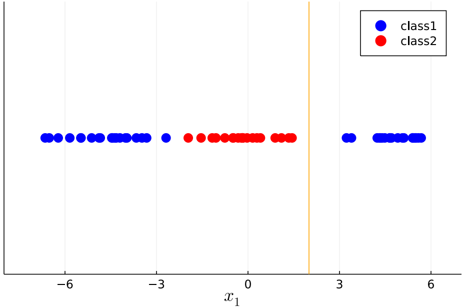
Computational form

- Linear: output y is a linear / affine function of input $\boldsymbol{x}$
- Nonlinear

Why nonlinearity? (Classification)

Example: supervised classifier learning

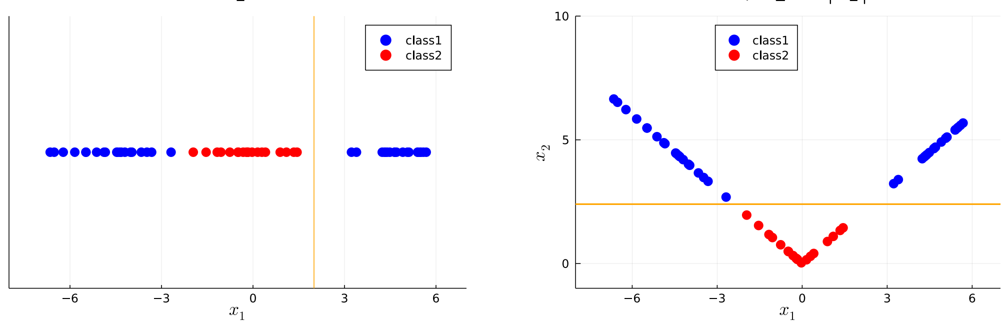
$$\boldsymbol{x} = x_1 \in \mathbb{R}$$



Why nonlinearity? (Classification)

Example: supervised classifier learning

$\boldsymbol{x} = x_1 \in \mathbb{R}$ $\qquad$ $\boldsymbol{x} \in \mathbb{R}^2, \; x_2 \triangleq |x_1|$



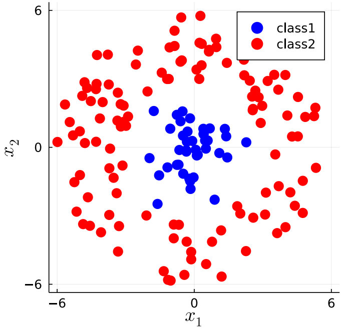
In this (simple, synthetic) example, nonlinear "lifting" from 1D to 2D enables a basic "linear" classifier from $(x_1, x_2) = (x_1, |x_1|)$.

(Inspired by `https://www.youtube.com/watch?v=3liCbRZPrZA`)

Why nonlinearity? (2D Classification case)

J. Fessler

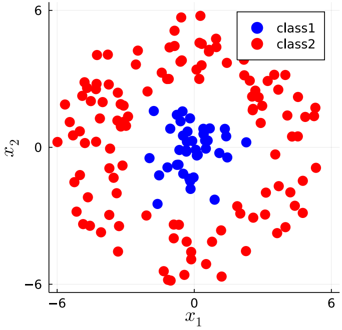
$$\boldsymbol{x} = (x_1, x_2) \in \mathbb{R}^2$$



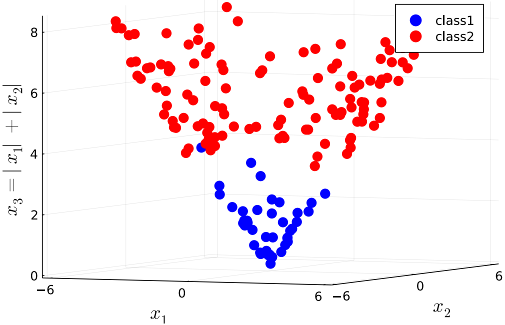
Why nonlinearity? (2D Classification case)

J. Fessler

$\boldsymbol{x} = (x_1, x_2) \in \mathbb{R}^2$

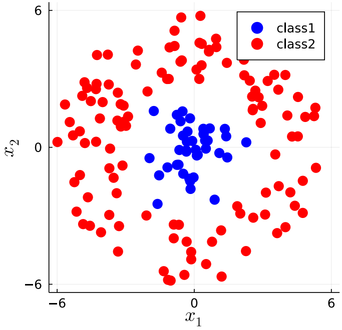


$\boldsymbol{x} \in \mathbb{R}^3,\ x_3 \triangleq |x_1| + |x_2|$

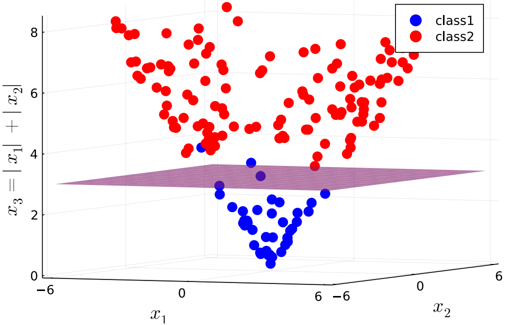


One additional nonlinear "feature" enables linear separation: $\boldsymbol{x} = (x_1, x_2, |x_1| + |x_2|)$

$\boldsymbol{x} = (x_1, x_2) \in \mathbb{R}^2$



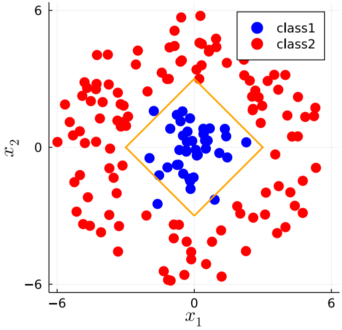
$\boldsymbol{x} \in \mathbb{R}^3,\ x_3 \triangleq |x_1| + |x_2|$



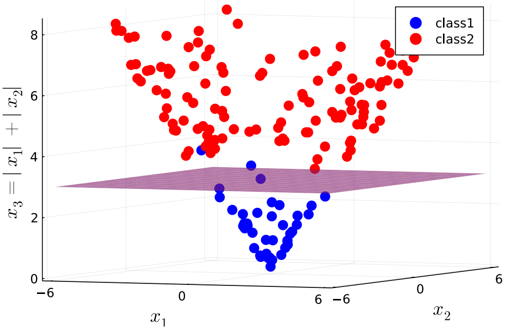
One additional nonlinear "feature" enables linear separation: $\boldsymbol{x} = (x_1, x_2, |x_1| + |x_2|)$

Why nonlinearity? (2D Classification case)

$\boldsymbol{x} = (x_1, x_2) \in \mathbb{R}^2$



$\boldsymbol{x} \in \mathbb{R}^3, \ x_3 \triangleq |x_1| + |x_2|$



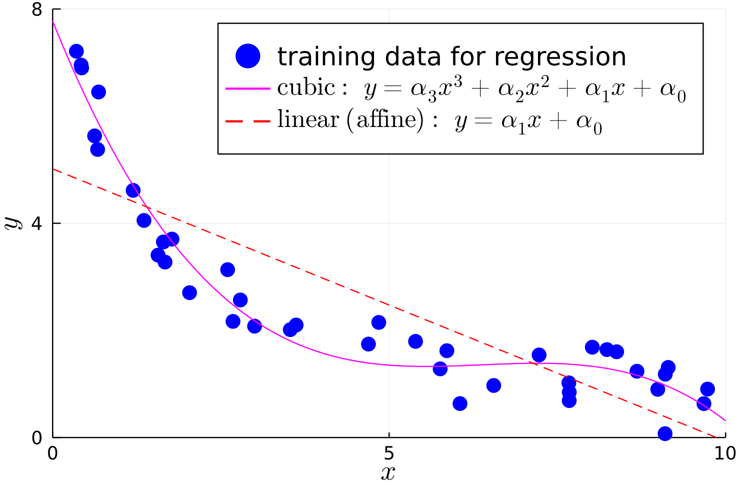
One additional nonlinear "feature" enables linear separation: $\boldsymbol{x} = (x_1, x_2, |x_1| + |x_2|)$

Many artificial neural nets (ANNs) use nonlinear rectified linear unit:

$\text{ReLU}(x) = \max(x, 0)$, where $|x| = \text{ReLU}(x) + \text{ReLU}(-x)$.

Why nonlinearity? (Regression)

J. Fessler

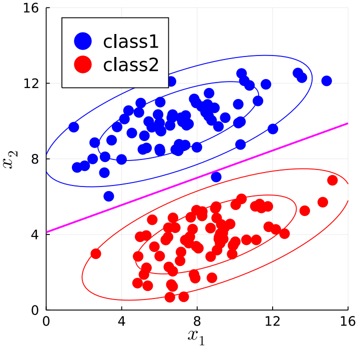


Assuming:
- Normal distributions
- Equal covariances

Optimal decision boundary is a line in 2D (hyperplane in general)

Optimal classifier is (mostly) linear:

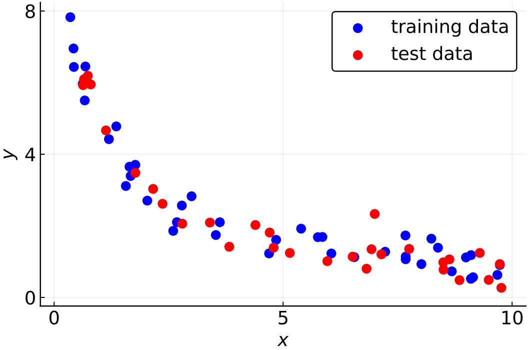
$$y = \begin{cases} \text{class1}, & \boldsymbol{w}'\boldsymbol{x} < \text{threshold} \\ \text{class2}, & \text{otherwise} \end{cases}$$



https://en.wikipedia.org/wiki/Linear_discriminant_analysis

Outline

- Most ML methods lack p-values, confidence intervals, Type I/II error formulae, ...
- Performance evaluation is performed *empirically* using testing data,
- after training the method ("learning") using training data.

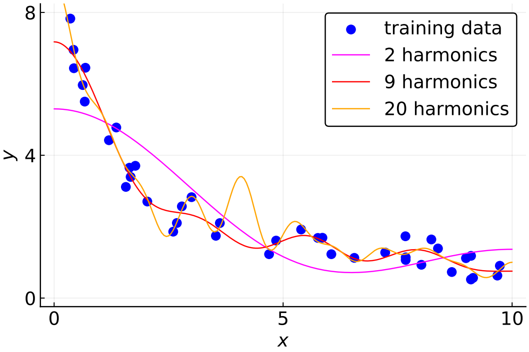


Model-order selection

ML methods have two categories of design choices:

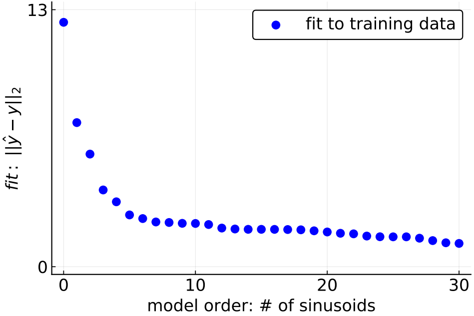
- Architecture / model order
- Tunable parameters (coefficients)

  We can learn the coefficients from training data for any given model order:



Training data: not for model selection

Fitting "error" with various numbers of sinusoids:

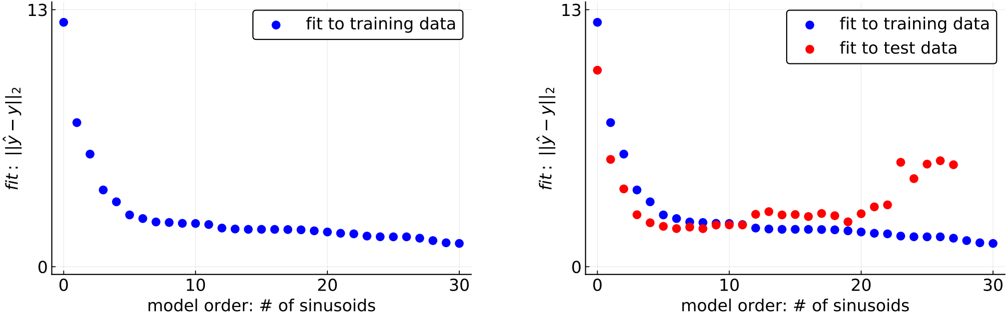


- More sinusoids (more degrees of freedom / larger model order) $\Longrightarrow$ "better" fit to the training data

Training data: not for model selection

J. Fessler

Fitting "error" with various numbers of sinusoids:

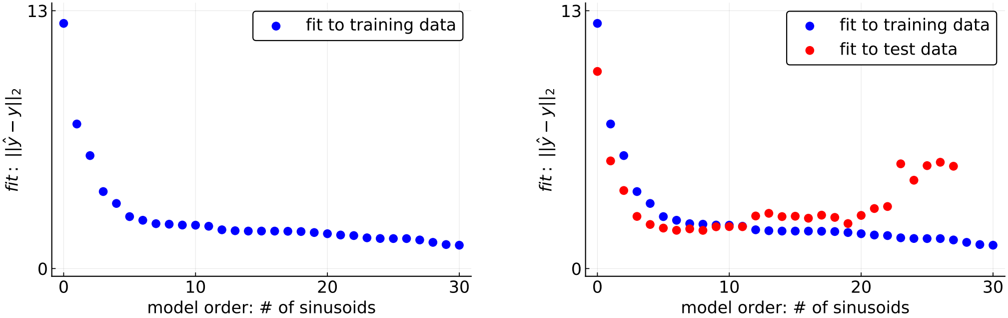


- More sinusoids (more degrees of freedom / larger model order) $\Longrightarrow$ "better" fit to the training data
- Over-fit if model order is "too high" $\Longrightarrow$ poor generalization / test results

Training data: not for model selection

J. Fessler

Fitting "error" with various numbers of sinusoids:

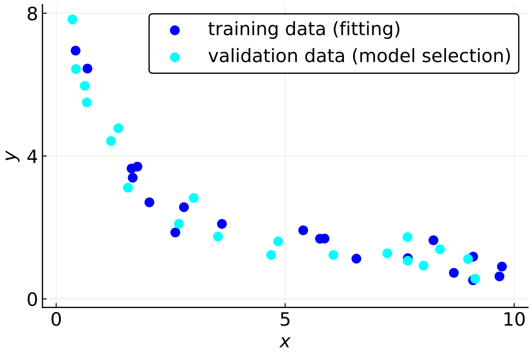


- More sinusoids (more degrees of freedom / larger model order) $\Longrightarrow$ "better" fit to the training data
- Over-fit if model order is "too high" $\Longrightarrow$ poor generalization / test results
- Cannot use the test data for training / model-order selection!

Validation data (e.g., cross validation)

Separate training data into two groups:

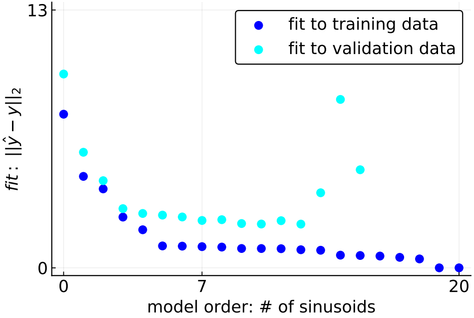
- training data
  for fitting parameters (coefficients)
- validation data
  for selecting model order / architecture



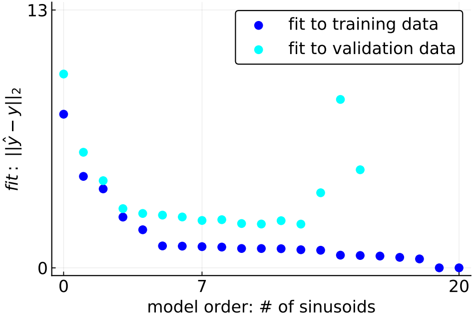
- (50-50% holdout shown here; one of many **cross validation** options)

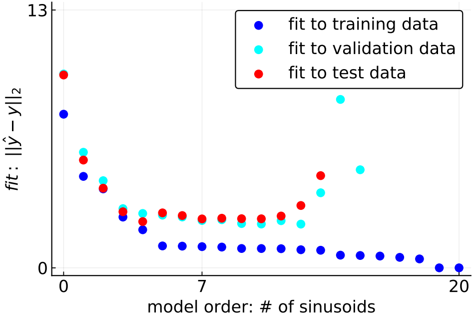
Validation data for model-order selection

J. Fessler

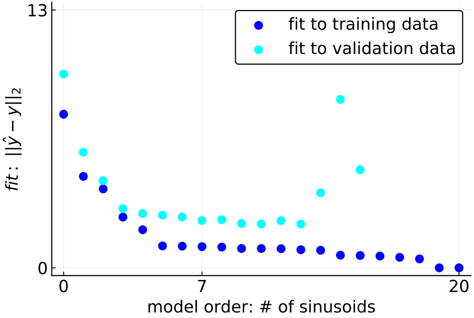


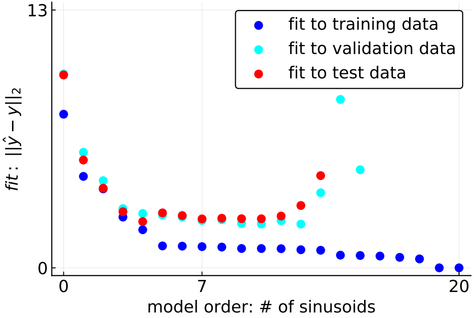
Validation data for model-order selection





Validation data for model-order selection





- Options for model-order selection:
  - Choose minimum of validation loss curve
  - Stop increasing model order when validation loss first increases (first sign of over-fitting)
- Attempts to assess how well the results will generalize to new data (red vs cyan)

Outline

Training an artificial neural network: overview

J. Fessler

Input: features $\rightarrow$ artificial NN $\rightarrow$ Output: prediction

Goal (supervised learning):
train NN so that output closely matches training data, without over fitting

(requires math...)

Training an artificial neural network: details

Input: features $\boldsymbol{x} \in \mathbb{R}^d$ $\rightarrow$ NN parameters $\boldsymbol{\theta}$ $\rightarrow$ Output: prediction: response (regression) $\hat{\boldsymbol{y}} = f(\boldsymbol{x}; \boldsymbol{\theta}) \in \mathbb{R}^m$

Training an artificial neural network: details

J. Fessler

Input: features $\boldsymbol{x} \in \mathbb{R}^d$ $\rightarrow$ NN parameters $\boldsymbol{\theta}$ $\rightarrow$ Output: prediction: response (regression) $\hat{\boldsymbol{y}} = f(\boldsymbol{x}; \boldsymbol{\theta}) \in \mathbb{R}^m$

- Supervised training problem: given training data $(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_N, \boldsymbol{y}_N)$, learn parameters $\boldsymbol{\theta}$ of NN so that $\hat{\boldsymbol{y}}_n \triangleq f(\boldsymbol{x}_n; \boldsymbol{\theta}) \approx \boldsymbol{y}_n$.

Training an artificial neural network: details

J. Fessler

Input: features $\boldsymbol{x} \in \mathbb{R}^d$ $\rightarrow$ NN parameters $\boldsymbol{\theta}$ $\rightarrow$ Output: prediction: response (regression) $\hat{\boldsymbol{y}} = f(\boldsymbol{x}; \boldsymbol{\theta}) \in \mathbb{R}^m$

- Supervised training problem: given training data $(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_N, \boldsymbol{y}_N)$, learn parameters $\boldsymbol{\theta}$ of NN so that $\hat{\boldsymbol{y}}_n \triangleq f(\boldsymbol{x}_n; \boldsymbol{\theta}) \approx \boldsymbol{y}_n$.
- Quantify "≈" using a loss function $\ell(\hat{\boldsymbol{y}}_n, \boldsymbol{y}_n)$ such as $\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \|\hat{\boldsymbol{y}} - \boldsymbol{y}\|_2^2$.

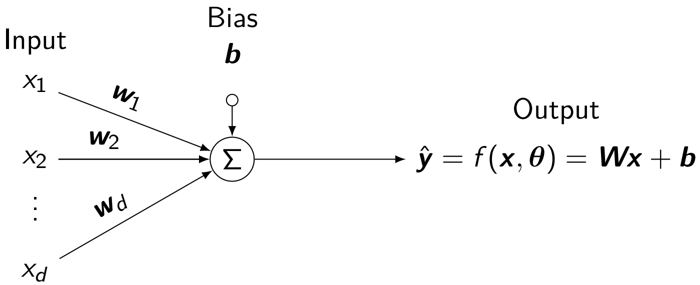
Training an artificial neural network: details

Input: features $\boldsymbol{x} \in \mathbb{R}^d$	$\to$	NN parameters $\boldsymbol{\theta}$	$\to$	Output: prediction: response (regression) $\hat{\boldsymbol{y}} = f(\boldsymbol{x}; \boldsymbol{\theta}) \in \mathbb{R}^m$

- Supervised training problem: given training data $(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_N, \boldsymbol{y}_N)$, learn parameters $\boldsymbol{\theta}$ of NN so that $\hat{\boldsymbol{y}}_n \triangleq f(\boldsymbol{x}_n; \boldsymbol{\theta}) \approx \boldsymbol{y}_n$.
- Quantify "$\approx$" using a loss function $\ell(\hat{\boldsymbol{y}}_n, \boldsymbol{y}_n)$ such as $\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \|\hat{\boldsymbol{y}} - \boldsymbol{y}\|_2^2$.
- Training is an optimization problem (minimize average loss):

$$\boldsymbol{\theta}_* = \arg\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{Y}), \qquad L(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{Y}) \triangleq \frac{1}{N} \sum_{n=1}^{N} \ell(f(\boldsymbol{x}_n; \boldsymbol{\theta}), \boldsymbol{y}_n).$$

Simplest example: affine NN (dense / fully connected)



- $\boldsymbol{x} \in \mathbb{R}^d$ is input
- $\boldsymbol{W} \in \mathbb{R}^{m \times d}$ are weights
- $\boldsymbol{b} \in \mathbb{R}^m$ is offset or bias
- $\boldsymbol{y} \in \mathbb{R}^m$ is output (response / prediction)
- NN parameters are weights and bias: $\boldsymbol{\theta} = (\boldsymbol{W}, \boldsymbol{b})$

Training an affine NN

Squared error loss: $\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \|\hat{\boldsymbol{y}} - \boldsymbol{y}\|_2^2 \implies$ training cost function is:

$$L(\theta; \boldsymbol{X}, \boldsymbol{Y}) = \left\| \begin{bmatrix} \boldsymbol{y}_1 & \dots & \boldsymbol{y}_N \end{bmatrix} - \boldsymbol{W} \begin{bmatrix} \boldsymbol{x}_1 & \dots & \boldsymbol{x}_N \end{bmatrix} - \boldsymbol{b}\mathbf{1}_N' \right\|_{\mathrm{F}}^2 .$$

Training an affine NN

Squared error loss: $\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \|\hat{\boldsymbol{y}} - \boldsymbol{y}\|_2^2 \implies$ training cost function is:

$$L(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{Y}) = \left\| \begin{bmatrix} \boldsymbol{y}_1 & \dots & \boldsymbol{y}_N \end{bmatrix} - \boldsymbol{W} \begin{bmatrix} \boldsymbol{x}_1 & \dots & \boldsymbol{x}_N \end{bmatrix} - \boldsymbol{b}\boldsymbol{1}_N' \right\|_{\mathrm{F}}^2 .$$

Optimization has analytical solution from $\nabla_{\boldsymbol{\theta}} L = \boldsymbol{0}$, leads to MMSE form:

$$\hat{\boldsymbol{y}} = f(\boldsymbol{x}, \boldsymbol{\theta}_*) = \boldsymbol{\mu}_y + \underbrace{\boldsymbol{K}_{yx} \boldsymbol{K}_x^{-1}}_{\boldsymbol{W}_*} (\boldsymbol{x} - \boldsymbol{\mu}_x), \quad \boldsymbol{\mu}_x = \frac{1}{N} \sum_{n=1}^N \boldsymbol{x}_n, \quad \boldsymbol{\mu}_y = \frac{1}{N} \sum_{n=1}^N \boldsymbol{y}_n,$$

$$\boldsymbol{K}_x = \frac{1}{N} \sum_{n=1}^N (\boldsymbol{x}_n - \boldsymbol{\mu}_x)(\boldsymbol{x}_n - \boldsymbol{\mu}_x)', \quad \boldsymbol{K}_{yx} = \frac{1}{N} \sum_{n=1}^N (\boldsymbol{y}_n - \boldsymbol{\mu}_y)(\boldsymbol{x}_n - \boldsymbol{\mu}_x)'.$$

Training an affine NN

J. Fessler

Squared error loss: $\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \|\hat{\boldsymbol{y}} - \boldsymbol{y}\|_2^2 \Longrightarrow$ training cost function is:

$$L(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{Y}) = \left\| \begin{bmatrix} \boldsymbol{y}_1 & \dots & \boldsymbol{y}_N \end{bmatrix} - \boldsymbol{W} \begin{bmatrix} \boldsymbol{x}_1 & \dots & \boldsymbol{x}_N \end{bmatrix} - \boldsymbol{b}\boldsymbol{1}_N' \right\|_{\mathrm{F}}^2 .$$

Optimization has analytical solution from $\nabla_{\boldsymbol{\theta}} L = \boldsymbol{0}$, leads to MMSE form:

$$\hat{\boldsymbol{y}} = f(\boldsymbol{x}, \boldsymbol{\theta}_*) = \boldsymbol{\mu}_y + \underbrace{\boldsymbol{K}_{yx} \, \boldsymbol{K}_x^{-1}}_{\boldsymbol{W}_*} (\boldsymbol{x} - \boldsymbol{\mu}_x), \quad \boldsymbol{\mu}_x = \frac{1}{N} \sum_{n=1}^N \boldsymbol{x}_n, \quad \boldsymbol{\mu}_y = \frac{1}{N} \sum_{n=1}^N \boldsymbol{y}_n,$$

$$\boldsymbol{K}_x = \frac{1}{N} \sum_{n=1}^N (\boldsymbol{x}_n - \boldsymbol{\mu}_x)(\boldsymbol{x}_n - \boldsymbol{\mu}_x)', \quad \boldsymbol{K}_{yx} = \frac{1}{N} \sum_{n=1}^N (\boldsymbol{y}_n - \boldsymbol{\mu}_y)(\boldsymbol{x}_n - \boldsymbol{\mu}_x)'.$$

- Need $N \geq d$ so that feature covariance matrix $\boldsymbol{K}_x$ is invertible (more training samples N than feature dimension d). Otherwise some regularization of weights is needed.

Training an affine NN

Squared error loss: $\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \|\hat{\boldsymbol{y}} - \boldsymbol{y}\|_2^2 \implies$ training cost function is:

$$L(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{Y}) = \left\| \begin{bmatrix} \boldsymbol{y}_1 & \dots & \boldsymbol{y}_N \end{bmatrix} - \boldsymbol{W} \begin{bmatrix} \boldsymbol{x}_1 & \dots & \boldsymbol{x}_N \end{bmatrix} - \boldsymbol{b}\boldsymbol{1}_N' \right\|_{\mathrm{F}}^2 .$$

Optimization has analytical solution from $\nabla_{\boldsymbol{\theta}} L = \boldsymbol{0}$, leads to MMSE form:

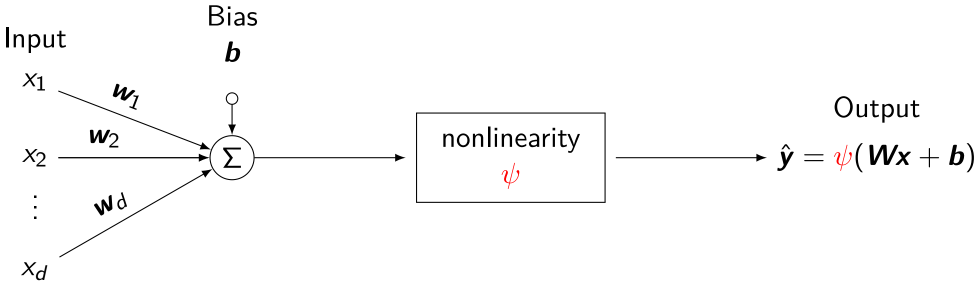
$$\hat{\boldsymbol{y}} = f(\boldsymbol{x}, \boldsymbol{\theta}_*) = \boldsymbol{\mu}_y + \underbrace{\boldsymbol{K}_{yx} \boldsymbol{K}_x^{-1}}_{\boldsymbol{W}_*} (\boldsymbol{x} - \boldsymbol{\mu}_x), \quad \boldsymbol{\mu}_x = \frac{1}{N} \sum_{n=1}^N \boldsymbol{x}_n, \quad \boldsymbol{\mu}_y = \frac{1}{N} \sum_{n=1}^N \boldsymbol{y}_n,$$

$$\boldsymbol{K}_x = \frac{1}{N} \sum_{n=1}^N (\boldsymbol{x}_n - \boldsymbol{\mu}_x)(\boldsymbol{x}_n - \boldsymbol{\mu}_x)', \quad \boldsymbol{K}_{yx} = \frac{1}{N} \sum_{n=1}^N (\boldsymbol{y}_n - \boldsymbol{\mu}_y)(\boldsymbol{x}_n - \boldsymbol{\mu}_x)'.$$

- Need $N \geq d$ so that feature covariance matrix $\boldsymbol{K}_x$ is invertible (more training samples N than feature dimension d). Otherwise some regularization of weights is needed.
- This simple case is one of very few with analytical (noniterative) solution for $\boldsymbol{\theta}_*$

Nonlinear artificial neuron

Perceptron: Rosenblatt, 1957 [2]



- No analytical solution for training NN parameters $\boldsymbol{W}, \boldsymbol{b}$
- Iterative methods required

Kernel ridge regression (nonlinearity)

$$\boldsymbol{x} \to \boxed{\begin{array}{c}\text{Nonlinear}\\ \text{function}\\ \phi : \mathbb{R}^d \mapsto \mathbb{R}^D\end{array}} \to \boldsymbol{z} \to \boxed{\begin{array}{c}\text{Affine}\\ \text{function}\\ \boldsymbol{W}\boldsymbol{z} + \boldsymbol{b}\end{array}} \to \hat{\boldsymbol{y}} = f(\boldsymbol{x}; \boldsymbol{\theta}) = \boldsymbol{W}\phi(\boldsymbol{x}) + \boldsymbol{b} \in \mathbb{R}^m$$

Kernel ridge regression (nonlinearity)

$$\boldsymbol{x} \to \boxed{\begin{array}{c}\text{Nonlinear} \\ \text{function} \\ \phi : \mathbb{R}^d \mapsto \mathbb{R}^D\end{array}} \to \boldsymbol{z} \to \boxed{\begin{array}{c}\text{Affine} \\ \text{function} \\ \boldsymbol{W}\boldsymbol{z} + \boldsymbol{b}\end{array}} \to \hat{\boldsymbol{y}} = f(\boldsymbol{x}; \boldsymbol{\theta}) = \boldsymbol{W}\phi(\boldsymbol{x}) + \boldsymbol{b} \in \mathbb{R}^m$$

For MSE training loss and fixed ϕ, MMSE estimator is

$$\hat{\boldsymbol{y}} = \boldsymbol{\mu}_y + \boldsymbol{K}_{yz}\, \boldsymbol{K}_z^{-1}\, (\boldsymbol{z} - \boldsymbol{\mu}_z) = \boldsymbol{\mu}_y + \boldsymbol{K}_{yz}\, \boldsymbol{K}_z^{-1}\, (\phi(\boldsymbol{x}) - \boldsymbol{\mu}_z), \quad \boldsymbol{\mu}_z = \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{z}_n,$$

$$\boldsymbol{z}_n \triangleq \phi(\boldsymbol{x}_n), \quad \boldsymbol{K}_z = \frac{1}{N} \sum_{n=1}^{N} (\boldsymbol{z}_n - \boldsymbol{\mu}_z)(\boldsymbol{z}_n - \boldsymbol{\mu}_z)', \quad \boldsymbol{K}_{yz} = \frac{1}{N} \sum_{n=1}^{N} (\boldsymbol{y}_n - \boldsymbol{\mu}_y)(\boldsymbol{z}_n - \boldsymbol{\mu}_z)'.$$

Kernel ridge regression (nonlinearity)

$$\boldsymbol{x} \to \boxed{\begin{array}{c}\text{Nonlinear} \\ \text{function} \\ \phi : \mathbb{R}^d \mapsto \mathbb{R}^D\end{array}} \to \boldsymbol{z} \to \boxed{\begin{array}{c}\text{Affine} \\ \text{function} \\ \boldsymbol{W}\boldsymbol{z} + \boldsymbol{b}\end{array}} \to \hat{\boldsymbol{y}} = f(\boldsymbol{x}; \boldsymbol{\theta}) = \boldsymbol{W}\phi(\boldsymbol{x}) + \boldsymbol{b} \in \mathbb{R}^m$$

For MSE training loss and fixed ϕ, MMSE estimator is

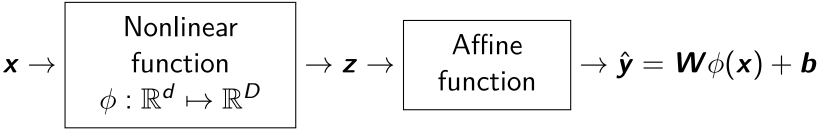
$$\hat{\boldsymbol{y}} = \boldsymbol{\mu}_y + \boldsymbol{K}_{yz}\, \boldsymbol{K}_z^{-1}\, (\boldsymbol{z} - \boldsymbol{\mu}_z) = \boldsymbol{\mu}_y + \boldsymbol{K}_{yz}\, \boldsymbol{K}_z^{-1}\, (\phi(\boldsymbol{x}) - \boldsymbol{\mu}_z), \quad \boldsymbol{\mu}_z = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{z}_n,$$

$$\boldsymbol{z}_n \triangleq \phi(\boldsymbol{x}_n), \quad \boldsymbol{K}_z = \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{z}_n - \boldsymbol{\mu}_z)(\boldsymbol{z}_n - \boldsymbol{\mu}_z)', \quad \boldsymbol{K}_{yz} = \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{y}_n - \boldsymbol{\mu}_y)(\boldsymbol{z}_n - \boldsymbol{\mu}_z)'.$$

- Typically $D = \dim(\boldsymbol{z}) \gg d = \dim(\boldsymbol{x})$, so even more samples N could be needed.
- Solution is to use ridge regression: replace $\boldsymbol{K}_z^{-1}$ with $(\boldsymbol{K}_z + \alpha \boldsymbol{I})^{-1}$; choose α by cross validation.

Kernel ridge regression universality

J. Fessler



- Affine function $\boldsymbol{W}\boldsymbol{z} + \boldsymbol{b}$ is same as a fully connected NN layer without nonlinearity.
- Choosing a nonlinear function ϕ based on a Gaussian kernel is universal:
  can approximate regular functions to arbitrary accuracy as N increases [3, 4] using:

$$\phi(\boldsymbol{x}) = \begin{bmatrix} \mathrm{e}^{-\|\boldsymbol{x}-\boldsymbol{x}_1\|_{\boldsymbol{\Lambda}}^2} & \dots & \mathrm{e}^{-\|\boldsymbol{x}-\boldsymbol{x}_N\|_{\boldsymbol{\Lambda}}^2} \end{bmatrix}^T.$$

- Training is very easy and fast because only free parameters are linear ones: $\boldsymbol{W}$ and $\boldsymbol{b}$
- Shallow learning
- Suitable for low-dimensional problems like parameter quantification.

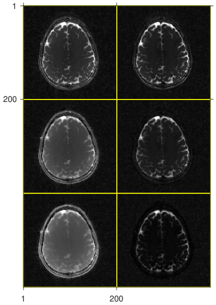
Quantitative MRI example

Quantitative MRI: images $\rightarrow$ estimation $\rightarrow$ parameters (T1,T2,. . .)

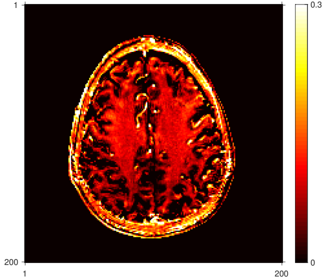
- Traditional nonlinear estimation methods:
  - nonlinear least squares
  - dictionary matching (quantized maximum likelihood via variable projection)
- Machine-learning methods
  - deep neural network regression [5–8]
    typically long training times
  - parameter estimation via kernel regression (PERK)
    Gopal Nataraj et al., ISBI 2017 [9], IEEE T-MI 2018 [3], arXiv 1809.08908 [10],

MWF PERK example

Myelin water fraction (MWF) estimated from 3 DESS scans
with optimized flip angles 33.0, 18.3, 15.1° and TRs 17.5, 30.2, 60.3 ms. [10–12]



→ PERK →



$$\text{Input} \quad \rightarrow \boxed{\text{NN with parameters } \boldsymbol{\theta}} \rightarrow \quad \text{Output}$$

Learning NN parameters (training) requires optimization (minimize average loss):

$$\boldsymbol{\theta}_* = \arg\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{Y}), \quad L(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{Y}) \triangleq \frac{1}{N} \sum_{n=1}^N \ell(f(\boldsymbol{x}_n; \boldsymbol{\theta}), \boldsymbol{y}_n)$$

Input $\rightarrow$ | NN with parameters $\boldsymbol{\theta}$ | $\rightarrow$ Output

Learning NN parameters (training) requires optimization (minimize average loss):

$$\boldsymbol{\theta}_* = \arg\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{Y}), \quad L(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{Y}) \triangleq \frac{1}{N} \sum_{n=1}^{N} \ell(f(\boldsymbol{x}_n; \boldsymbol{\theta}), \boldsymbol{y}_n)$$

- Cannot solve $\nabla_{\boldsymbol{\theta}} L = \mathbf{0}$ analytically in general.

Training as an optimization problem

J. Fessler

Input $\rightarrow$ | NN with parameters $\boldsymbol{\theta}$ | $\rightarrow$ Output

Learning NN parameters (training) requires optimization (minimize average loss):

$$\boldsymbol{\theta}_* = \arg\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{Y}), \quad L(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{Y}) \triangleq \frac{1}{N} \sum_{n=1}^{N} \ell(f(\boldsymbol{x}_n; \boldsymbol{\theta}), \boldsymbol{y}_n)$$

- Cannot solve $\nabla_{\boldsymbol{\theta}} L = \mathbf{0}$ analytically in general.
- Natural approach is (slow!) gradient descent iteration for $k = 0, 1, \ldots$

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_k),$$

  - step size $\alpha > 0$ aka "learning rate"
  - the gradient $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_k)$ is the vector of partial derivatives of the loss function w.r.t. every NN parameter.
  - Initializer $\boldsymbol{\theta}_0$ often random

Accelerating training

- Use mini-batch approximation to gradient of loss:

$$\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_k) = \underbrace{\frac{1}{N}\sum_{n=1}^{N}}_{\text{all data}} \nabla_{\boldsymbol{\theta}} \ell(f(\boldsymbol{x}_n; \boldsymbol{\theta}_k), \boldsymbol{y}_n) \approx \underbrace{\frac{1}{|\mathcal{S}_k|}\sum_{n \in \mathcal{S}_k}}_{\text{some data}} \nabla_{\boldsymbol{\theta}} \ell(f(\boldsymbol{x}_n; \boldsymbol{\theta}_k), \boldsymbol{y}_n),$$

where $\mathcal{S}_k$ is a (often random) subset of the data at kth iteration.
  - Mini-batch size often matched to # of compute threads.
  - Aka **stochastic gradient descent** (SGD) or incremental gradients.

Accelerating training

- Use mini-batch approximation to gradient of loss:

$$\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_k) = \underbrace{\frac{1}{N} \sum_{n=1}^{N}}_{\text{all data}} \nabla_{\boldsymbol{\theta}} \ell(f(\boldsymbol{x}_n; \boldsymbol{\theta}_k), \boldsymbol{y}_n) \approx \underbrace{\frac{1}{|\mathcal{S}_k|} \sum_{n \in \mathcal{S}_k}}_{\text{some data}} \nabla_{\boldsymbol{\theta}} \ell(f(\boldsymbol{x}_n; \boldsymbol{\theta}_k), \boldsymbol{y}_n),$$

  where $\mathcal{S}_k$ is a (often random) subset of the data at kth iteration.
  - Mini-batch size often matched to # of compute threads.
  - Aka **stochastic gradient descent** (SGD) or incremental gradients.
- Momentum
- Automated step-size selection [13]
- Use GPUs...

The gradient operation looks simple on paper:

$$\nabla_{\boldsymbol{\theta}} \ell(f(\boldsymbol{x}; \boldsymbol{\theta}), \boldsymbol{y}) = \begin{bmatrix} \frac{\partial}{\partial \theta_1} \ell(f(\boldsymbol{x}; \boldsymbol{\theta}), \boldsymbol{y}) \\ \vdots \\ \frac{\partial}{\partial \theta_K} \ell(f(\boldsymbol{x}; \boldsymbol{\theta}), \boldsymbol{y}) \end{bmatrix},$$

but for deep networks the model is a cascade of many functions, one per layer:

$$\boldsymbol{x} \to \boxed{f_1(\cdot; \boldsymbol{\theta})} \to \boxed{f_2(\cdot; \boldsymbol{\theta})} \to \cdots \to \boxed{f_L(\cdot; \boldsymbol{\theta})} \to f(\boldsymbol{x}; \boldsymbol{\theta}) = f_L(\cdots f_2(f_1(\boldsymbol{x}; \boldsymbol{\theta}); \boldsymbol{\theta}); \boldsymbol{\theta}).$$

- In practice most layers have different parameters, but some parameters may affect multiple layers (especially RNN)

Backpropagation

The gradient operation looks simple on paper:

$$\nabla_{\boldsymbol{\theta}} \ell(f(\boldsymbol{x};\boldsymbol{\theta}),\boldsymbol{y}) = \begin{bmatrix} \frac{\partial}{\partial \theta_1} \ell(f(\boldsymbol{x};\boldsymbol{\theta}),\boldsymbol{y}) \\ \vdots \\ \frac{\partial}{\partial \theta_K} \ell(f(\boldsymbol{x};\boldsymbol{\theta}),\boldsymbol{y}) \end{bmatrix},$$

but for deep networks the model is a cascade of many functions, one per layer:

$$\boldsymbol{x} \to \boxed{f_1(\cdot;\boldsymbol{\theta})} \to \boxed{f_2(\cdot;\boldsymbol{\theta})} \to \cdots \to \boxed{f_L(\cdot;\boldsymbol{\theta})} \to f(\boldsymbol{x};\boldsymbol{\theta}) = f_L(\cdots f_2(f_1(\boldsymbol{x};\boldsymbol{\theta});\boldsymbol{\theta});\boldsymbol{\theta}).$$

- In practice most layers have different parameters, but some parameters may affect multiple layers (especially RNN)
- Backpropagation = chain rule for differentiation, hopefully efficiently coded [14] [15]
- Convenient software tools provide automatic differentiation (Python: TensorFlow, PyTorch, ...) (Julia: Flux, ...) (Matlab: MatConvNet?)

Backpropagation illustration (1)

Consider a two-layer NN with a single weight to be learned in the first layer:

$$\begin{array}{c} \text{Input} \\ x \end{array} \to \boxed{\begin{array}{c} \text{Layer1} \\ h_w(\cdot) \end{array}} \xrightarrow{h_w(x)} \boxed{\begin{array}{c} \text{Layer2} \\ g(\cdot) \end{array}} \xrightarrow{g(h_w(x))} \begin{array}{c} \text{Output} \\ \hat{y} = g(h_w(x)) \end{array} \to \boxed{\begin{array}{c} \text{Loss} \\ L(w) \end{array}}$$

Backpropagation illustration (1)

Consider a two-layer NN with a single weight to be learned in the first layer:

$$\begin{array}{c} \text{Input} \\ x \end{array} \rightarrow \boxed{\begin{array}{c} \text{Layer1} \\ h_w(\cdot) \end{array}} \xrightarrow{h_w(x)} \boxed{\begin{array}{c} \text{Layer2} \\ g(\cdot) \end{array}} \xrightarrow{g(h_w(x))} \begin{array}{c} \text{Output} \\ \hat{y} = g(h_w(x)) \end{array} \rightarrow \boxed{\begin{array}{c} \text{Loss} \\ L(w) \end{array}}$$

Loss function for a single training sample:

$$L(w) = \ell(g(h_w(x)), y).$$

Consider a two-layer NN with a single weight to be learned in the first layer:

$$\underset{x}{\text{Input}} \rightarrow \boxed{\underset{h_w(\cdot)}{\text{Layer1}}} \xrightarrow{h_w(x)} \boxed{\underset{g(\cdot)}{\text{Layer2}}} \xrightarrow{g(h_w(x))} \underset{\hat{y} = g(h_w(x))}{\text{Output}} \rightarrow \boxed{\underset{L(w)}{\text{Loss}}}$$

Loss function for a single training sample:

$$L(w) = \ell(g(h_w(x)), y).$$

Chain rule for derivative of loss w.r.t. weight w:

$$\frac{\partial}{\partial w} L(w) = \dot{L}(w) = \frac{\partial}{\partial w} \ell(f_w(x), y) = \dot{\ell}(g(h_w(x)), y) \, \dot{g}(h_w(x)) \, \dot{h}_w(x).$$

Two key ingredients two compute:

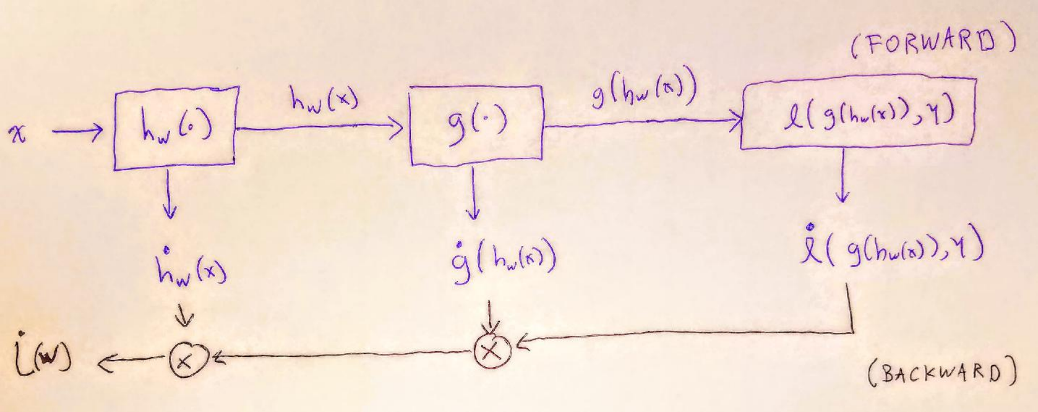
- Model at each layer of NN
- Derivatives of model at each layer, evaluated at layer input

Backpropagation illustration (2)

$$\dot{L}(w) = \dot{\ell}(g(h_w(x)), y) \quad \dot{g}(h_w(x)) \quad \dot{h}_w(x)$$
$$= \dot{h}_w(x) \quad \dot{g}(h_w(x)) \quad \dot{\ell}(g(h_w(x)), y).$$

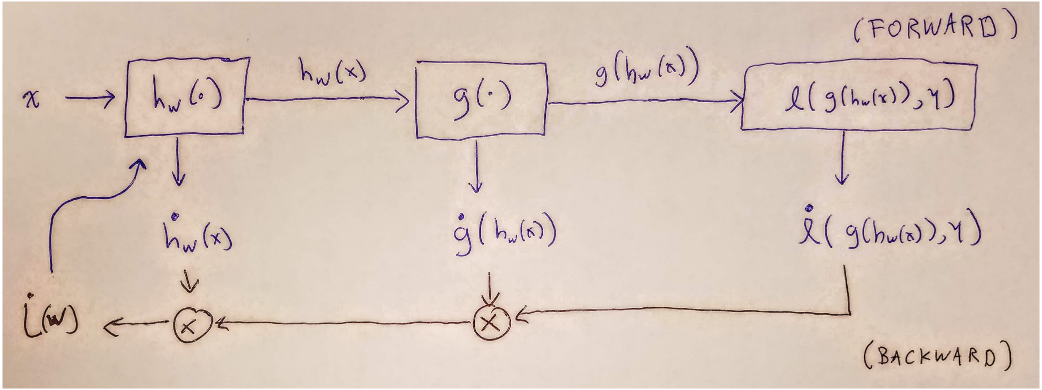
Backpropagation illustration (2)

$$\dot{L}(w) = \dot{\ell}(g(h_w(x)), y) \quad \dot{g}(h_w(x)) \quad \dot{h}_w(x)$$
$$= \dot{h}_w(x) \quad \dot{g}(h_w(x)) \quad \dot{\ell}(g(h_w(x)), y).$$

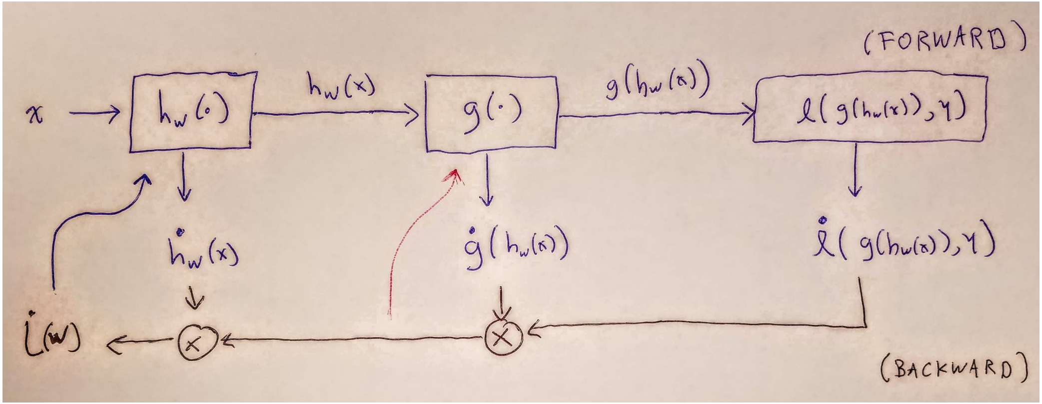


Backpropagation illustration (2)

$$\dot{L}(w) = \dot{\ell}(g(h_w(x)), y) \quad \dot{g}(h_w(x)) \quad \dot{h}_w(x)$$
$$= \dot{h}_w(x) \quad \dot{g}(h_w(x)) \quad \dot{\ell}(g(h_w(x)), y).$$

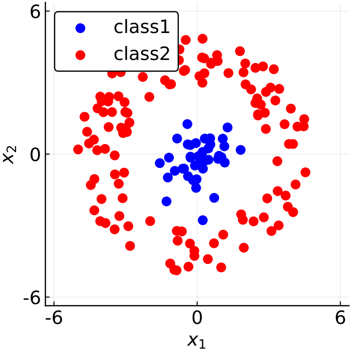


$$\dot{L}(w) = \dot{\ell}(g(h_w(x)), y) \quad \dot{g}(h_w(x)) \quad \dot{h}_w(x)$$
$$= \dot{h}_w(x) \quad \dot{g}(h_w(x)) \quad \dot{\ell}(g(h_w(x)), y).$$



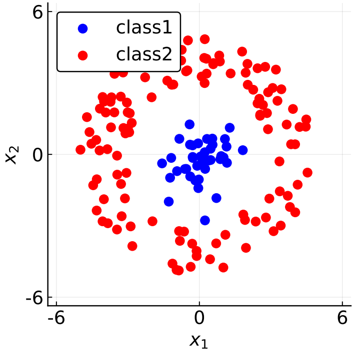
Supervised NN training example: binary classification

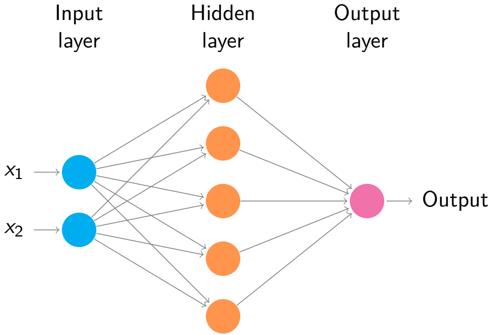
J. Fessler



- Nonlinearity is essential here

Supervised NN training example: binary classification





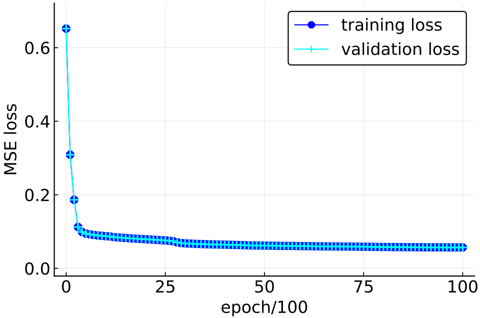
- Nonlinearity is essential here
- Each hidden node is a perceptron with $\text{ReLU}(x) = \max(x, 0)$
- Train output to be 1 for class2 and -1 for class1.

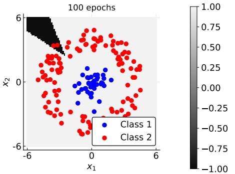
Example Flux code

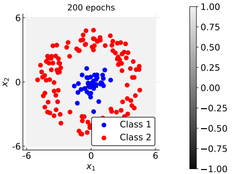
- Julia's Flux library [16] `http://fluxml.ai/Flux.jl`
- ML ingredients: training data $(\boldsymbol{X}, \boldsymbol{Y})$, model/architecture, loss function, optimizer
- For full Jupyter notebook see `https://tinyurl.com/ml2-18-jf`

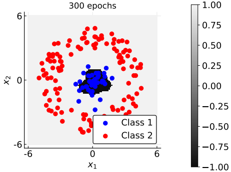
```
nhidden = 10 # neurons in hidden layer
model = Chain(Dense(2,nhidden,relu), Dense(nhidden,1)) # NN arch
loss(x, y) = mse(model(x), y)
iters = 10000 # hand crafted...
dataset = Base.Iterators.repeated((X, Y), iters)
Flux.train!(loss, dataset, ADAM(params(model)))
```

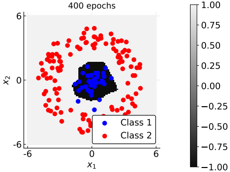
Flux NN training

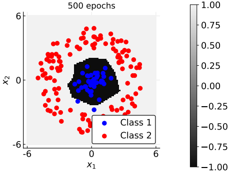


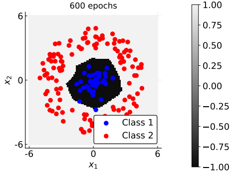








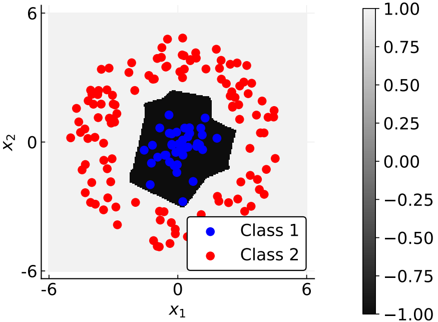




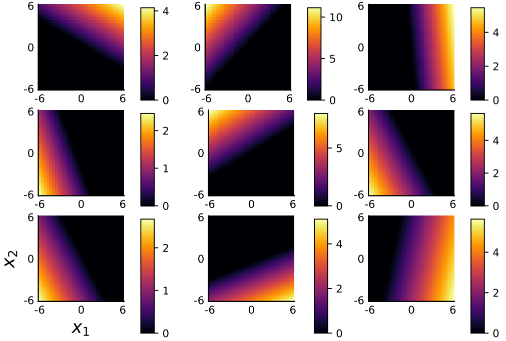
Flux results for binary classification

J. Fessler

Classifier results



Hidden layer functions



Principles generalize from binary classification to multiclass problems.

NN-based denoising example

See `https://tinyurl.com/ml2-18-jf`

Outline

Roles for machine learning in medical imaging

- Image analysis (post-processing):
  - classification: diagnosis / segmentation / treatment planning, ...
  - regression: localization / registration / quantification, ...
    (object size, e.g., vessel diameter, contrast concentration, T1, T2, ...)
- Image reconstruction
- Image acquisition

Machine learning in medical image interpretation

Most obvious place for machine learning is post-processing:

reconstructed images $\hat{\boldsymbol{x}}$ → ML-based image processing (classification or regression) → interpretation

Special issue of IEEE Trans. on Med. Imaging, May 2016 [17]

IEEE TRANSACTIONS ON MEDICAL IMAGING, VOL. 35, NO. 5, MAY 2016 1153

Guest Editorial

Deep Learning in Medical Imaging: Overview and Future Promise of an Exciting New Technique raw data $\boldsymbol{y}$ → ML-based image reconstruction → images $\hat{\boldsymbol{x}}$

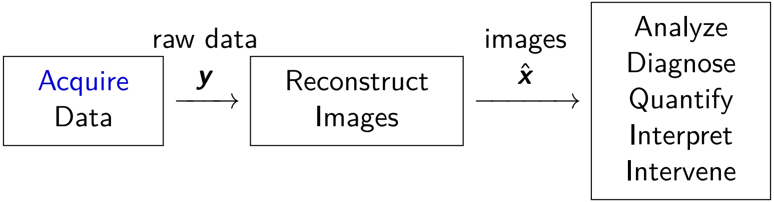
Special issue of IEEE Trans. on Medical Imaging, June 2018 [18]

IEEE TRANSACTIONS ON MEDICAL IMAGING, VOL. 37, NO. 6, JUNE 2018 1289

Image Reconstruction Is a New Frontier of Machine Learning

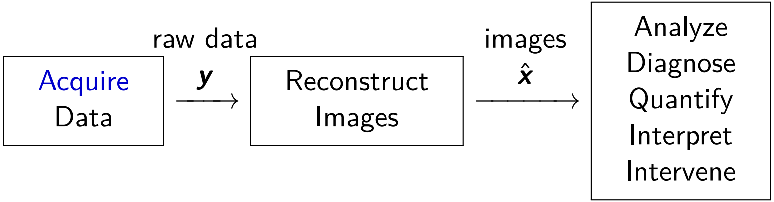
Ge Wang, *Fellow, IEEE*, Jong Chu Ye, *Senior Member, IEEE*, Klaus Mueller, *Senior Member, IEEE*, and Jeffrey A. Fessler, *Fellow, IEEE*

Machine learning in medical imaging: scan design



- Choose best k-space phase encoding locations based on training images:
  - "Learning-based compressive MRI" [19, 20]
    (Volkan Cevher group, June 2018 IEEE T-MI)
  - Yue Cao and David Levin, MRM Sep. 1993 "Feature recognizing MRI" [21–23]

Machine learning in medical imaging: scan design



- Choose best k-space phase encoding locations based on training images:
  - "Learning-based compressive MRI" [19, 20]
    (Volkan Cevher group, June 2018 IEEE T-MI)
  - Yue Cao and David Levin, MRM Sep. 1993 "Feature recognizing MRI" [21–23]
- Process fMRI data in real time, provide brain-state feedback to subject [24, 25]

Recommended reading (incomplete lists)

- Machine learning books: [26] [27] [28] [29] [30] [31] [32] [33]
- Survey paper(s) [34]
- Optimization: [35]
- DL overviews: [36–38]
- Generative models: [39, 40]:
- Deep learning myths [41]
- NN complexity analysis / function approximation [42–44] [45]
- Application to MR fingerprinting [5, 8]
- MR reconstruction / enhancement using CNN [46–54]
- Dynamic MR reconstruction using CNN [55]
- ...

Talk and code available online at
`https://tinyurl.com/ml2-18-jf`

Bibliography I

[1] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. “Julia: A fresh approach to numerical computing.” In: *SIAM Review* 59.1 (2017), 65–98.

[2] F. Rosenblatt. *The Perceptron - a perceiving and recognizing automaton*. Tech. rep. 85-460-1. Cornell: Aeronautical Laboratory, Jan. 1957.

[3] G. Nataraj, J-F. Nielsen, C. D. Scott, and J. A. Fessler. “Dictionary-free MRI PERK: Parameter estimation via regression with kernels.” In: *IEEE Trans. Med. Imag.* 37.9 (Sept. 2018), 2103–14.

[4] I. Steinwart and A. Christmann. *Support vector machines*. Springer, 2008.

[5] P. Virtue, S. X. Yu, and M. Lustig. “Better than real: Complex-valued neural nets for MRI fingerprinting.” In: *Proc. IEEE Intl. Conf. on Image Processing*. 2017, 3953–7.

[6] A. Lahiri, J. A. Fessler, and L. Hernandez-Garcia. “Optimized design of MRF scan parameters for ASL signal acquisition.” In: *ISMRM Workshop on MR Fingerprinting*. 2017.

[7] A. Lahiri, J. A. Fessler, and L. Hernandez-Garcia. “Optimized scan design for ASL fingerprinting and multiparametric estimation using neural network regression.” In: *Proc. Intl. Soc. Mag. Res. Med.* 2018, p. 309.

[8] O. Cohen, B. Zhu, and M. S. Rosen. “MR fingerprinting Deep RecOnstruction NEtwork (DRONE).” In: *Mag. Res. Med.* 80.3 (Sept. 2018), 885–94.

[9] G. Nataraj, J-F. Nielsen, and J. A. Fessler. “Dictionary-free MRI parameter estimation via kernel ridge regression.” In: *Proc. IEEE Intl. Symp. Biomed. Imag.* 2017, 5–9.

[10] G. Nataraj, J-F. Nielsen, M. Gao, and J. A. Fessler. *Fast, precise myelin water quantification using DESS MRI and kernel learning*. Submitted. 2018.

[11] G. Nataraj, M. Gao, J-F. Nielsen, and J. A. Fessler. “Kernel regression for fast myelin water imaging.” In: *ISMRM Workshop on Machine Learning Part 2*. 2018, p. 65.

[12] G. Nataraj, J-F. Nielsen, M. Gao, and J. A. Fessler. "Fast, precise myelin water quantification using DESS MRI and kernel learning." In: *Mag. Res. Med.* (2018). Submitted.

[13] D. P. Kingma and J. Ba. *Adam: A method for stochastic optimization*. 2014.

[14] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning representations by back-propagating errors." In: *Nature* 323.6088 (Oct. 1986), 533–6.

[15] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. "Backpropagation applied to handwritten zip code recognition." In: *Neural Computation* 1.4 (Dec. 1989), 541–51.

[16] M. Innes. "Flux: Elegant machine learning with Julia." In: *J. of Open Source Software* 3.25 (2018), p. 602.

[17] H. Greenspan, B. van Ginneken, and R. M. Summers. "Guest editorial deep learning in medical imaging: overview and future promise of an exciting new technique." In: *IEEE Trans. Med. Imag.* 35.5 (May 2016), 1153–9.

[18] G. Wang, J. C. Ye, K. Mueller, and J. A. Fessler. "Image reconstruction is a new frontier of machine learning." In: *IEEE Trans. Med. Imag.* 37.6 (June 2018), 1289–96.

[19] L. Baldassarre, Y-H. Li, J. Scarlett, B. Gozcu, I. Bogunovic, and V. Cevher. "Learning-based compressive subsampling." In: *IEEE J. Sel. Top. Sig. Proc.* 10.4 (June 2016), 809–22.

[20] B. Gozcu, R. K. Mahabadi, Y-H. Li, E. Ilicak, T. Cukur, J. Scarlett, and V. Cevher. "Learning-based compressive MRI." In: *IEEE Trans. Med. Imag.* 37.6 (June 2018), 1394–406.

[21] Y. Cao and D. N. Levin. "Feature-recognizing MRI." In: *Mag. Res. Med.* 30.3 (Sept. 1993), 305–17.

[22] Y. Cao, D. N. Levin, and L. Yao. "Locally focused MRI." In: *Mag. Res. Med.* 34.6 (Dec. 1995), 858–67.

[23] Y. Cao and D. N. Levin. "Using an image database to constrain the acquisition and reconstruction of MR images of the human head." In: *IEEE Trans. Med. Imag.* 14.2 (June 1995), 350–61.

[24] S. M. LaConte, S. J. Peltier, and X. P. Hu. "Real-time fMRI using brain-state classification." In: *Hum. Brain Map.* 28.10 (Oct. 2007), 1033–4.

[25] T. Watanabe, Y. Sasaki, K. Shibata, and M. Kawato. "Advances in fMRI Real-Time Neurofeedback." In: *Trends in Cognitive Sciences* 21.12 (Dec. 2017), 997–1010.

[26] K. Mardia, J. Kent, and J. Bibby. *Multivariate analysis*. Academic Press, 1979.

[27] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern classification*. New York: Wiley, 2001.

[28] B. Scholkopf and S. Smola. *Learning with kernels*. MIT, 2002.

[29] C. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[30] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning*. Springer, 2009.

[31] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of machine learning*. 2nd edition. MIT, 2018.

[32] K. P. Murphy. *Machine learning: A probabilistic perspective*. MIT, 2012.

[33] S. Shalev-Shwartz and S. Ben-David. *Understanding machine learning: from theory to algorithms*. Cambridge, 2014.

[34] O. Simeone. "A brief introduction to machine learning for engineers." In: *Found. & Trends in Sig. Pro.* 12.3-4 (2018), 200–431.

[35] S. Boyd and L. Vandenberghe. *Convex optimization*. UK: Cambridge, 2004.

[36] G. Wang. "A perspective on deep imaging." In: *IEEE Access* 4 (Nov. 2016), 8914–24.

[37] G. Wang, M. Kalra, and C. G. Orton. "Machine learning will transform radiology significantly within the next five years." In: *Med. Phys.* 44.6 (June 2017), 2041–4.

[38] M. T. McCann, K. H. Jin, and M. Unser. "Convolutional neural networks for inverse problems in imaging: A review." In: *IEEE Sig. Proc. Mag.* 34.6 (Nov. 2017), 85–95.

Bibliography IV

[39] I. Deshpande, Z. Zhang, and A. Schwing. "Generative modeling using the sliced Wasserstein distance." In: *Proc. IEEE Conf. on Comp. Vision and Pattern Recognition*. 2018.

[40] S. Kolouri, P. E. Pope, C. E. Martin, and G. K. Rohde. *Sliced-Wasserstein autoencoder: an embarrassingly simple generative model*. 2018.

[41] S. Rakhlin. *MythBusters: A Deep Learning Edition*. Slides dated Jan 18-19, 2018. 2018.

[42] N. Golowich, A. Rakhlin, and O. Shamir. *Size-independent sample complexity of neural networks*. 2017.

[43] T. Liang, T. Poggio, A. Rakhlin, and J. Stokes. *Fisher-Rao metric, geometry, and complexity of neural networks*. 2017.

[44] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein. "On the expressive power of deep neural networks." In: *Proc. Intl. Conf. Mach. Learn.* Vol. 70. 2017, 2847–54.

[45] S. Liang and R. Srikant. "Why deep neural networks for function approximation?" In: *Proc. Intl. Conf. on Learning Representations*. 2017.

[46] S. Ravishankar, I. Y. Chun, and J. A. Fessler. "Physics-driven deep training of dictionary-based algorithms for MR image reconstruction." In: *Proc., IEEE Asilomar Conf. on Signals, Systems, and Comp.* Invited. 2017, 1859–63.

[47] M. Mardani, E. Gong, J. Y. Cheng, S. S. Vasanawala, G. Zaharchuk, L. Xing, and J. M. Pauly. "Deep generative adversarial neural networks for compressive sensing MRI." In: *IEEE Trans. Med. Imag.* 38.1 (Jan. 2019), 167–79.

[48] K. Hammernik, T. Klatzer, E. Kobler, M. P. Recht, D. K. Sodickson, T. Pock, and F. Knoll. "Learning a variational network for reconstruction of accelerated MRI data." In: *Mag. Res. Med.* 79.6 (June 2018), 3055–71.

[49] B. Zhu, J. Z. Liu, S. F. Cauley, B. R. Rosen, and M. S. Rosen. "Image reconstruction by domain-transform manifold learning." In: *Nature* 555 (Mar. 2018), 487–92.

[50] Y. Han, J. Yoo, H. H. Kim, H. J. Shin, K. Sung, and J. C. Ye. "Deep learning with domain adaptation for accelerated projection-reconstruction MR." In: *Mag. Res. Med.* 80.3 (Sept. 2018), 1189–205.

[51] K. H. Jin and M. Unser. "3D BPConvNet to reconstruct parallel MRI." In: *Proc. IEEE Intl. Symp. Biomed. Imag.* 2018, 361–4.

[52] H. Jeelani, J. Martin, F. Vasquez, M. Salerno, and D. S. Weller. "Image quality affects deep learning reconstruction of MRI." In: *Proc. IEEE Intl. Symp. Biomed. Imag.* 2018, 357–60.

[53] T. M. Quan, T. Nguyen-Duc, and W-K. Jeong. "Compressed sensing MRI reconstruction using a generative adversarial network with a cyclic loss." In: *IEEE Trans. Med. Imag.* 37.6 (June 2018), 1488–97.

[54] T. Eo, Y. Jun, T. Kim, J. Jang, H-J. Lee, and D. Hwang. "KIKI-net: cross-domain convolutional neural networks for reconstructing undersampled magnetic resonance images." In: *Mag. Res. Med.* 80.5 (Nov. 2018), 2188–201.

[55] J. Schlemper, J. Caballero, J. V. Hajnal, A. N. Price, and D. Rueckert. "A deep cascade of convolutional neural networks for dynamic MR image reconstruction." In: *IEEE Trans. Med. Imag.* 37.2 (Feb. 2018), 491–503.