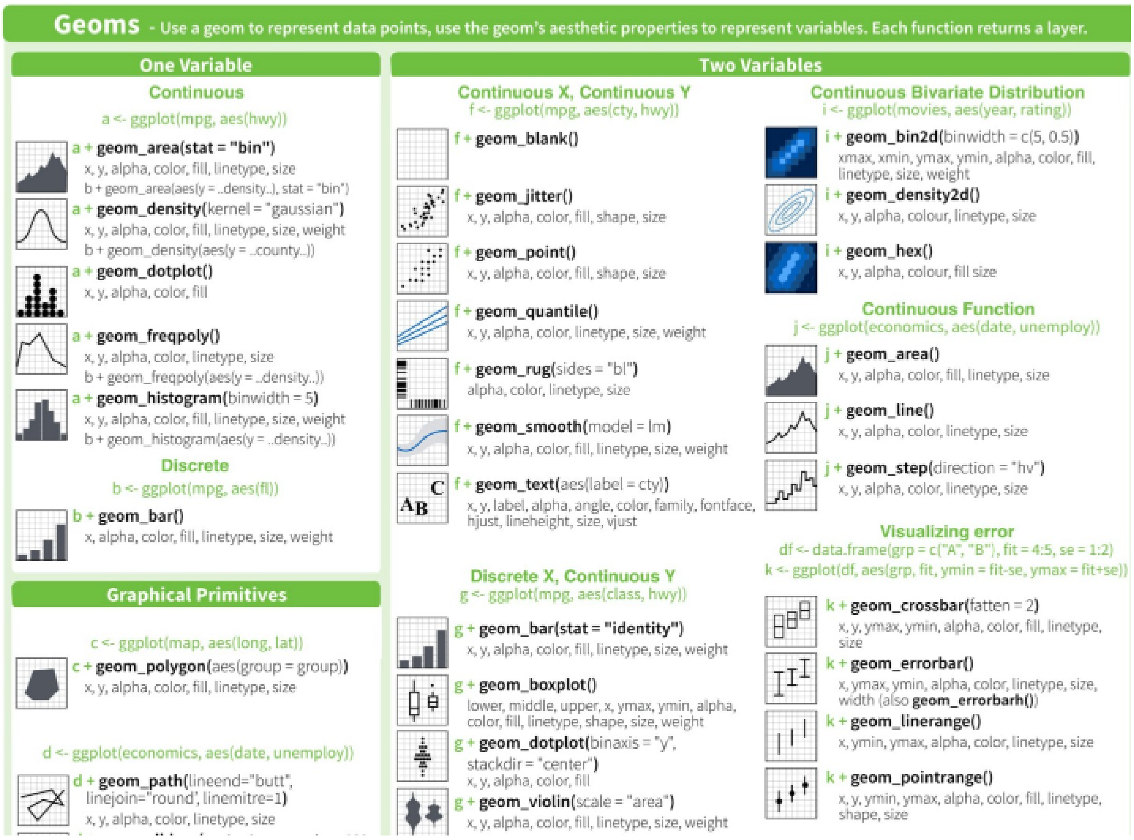
Unknown date | Unknown author

Large Collection of Neural Nets, Numpy, Pandas, Matplotlib, Scikit and ML Cheat Sheets

This collection covers much more than the topics listed in the title. It also features Azure, Python, Tensorflow, data visualization, and many other cheat sheets. Additional cheat sheets can be found here and here. Below is a screenshot (extract from the data visualization cheat sheet.)



The one below is rather interesting too, but the source is unknown, and anywhere it was posted, it is unreadable. This is the best rendering after 30 minutes of work:

General Minimization Algorithm:

$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$ *or* $\Delta\mathbf{x}_k = (\mathbf{x}_{k+1} - \mathbf{x}_k) = \alpha_k \mathbf{p}_k$

Steepest Descent Algorithm:

$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{g}_k$ where, $\mathbf{g}_k = \nabla F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}_k}$

Stable Learning Rate: ($\alpha_k = \alpha$, constant) $\alpha < \frac{2}{\lambda_{max}}$

$\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ Eigenvalues of Hessian matrix A

Learning Rate to Minimize Along the Line:

$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \overset{ls}{\Rightarrow} \alpha_k = -\frac{\mathbf{g}_k^T \mathbf{p}_k}{\mathbf{p}_k^T A \mathbf{p}_k}$ (For quadratic fn.)

After Minimization Along the Line:

$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \Rightarrow \mathbf{g}_{k+1}^T \mathbf{p}_k = 0$

ADALINE: $\mathbf{a} = purelin(\mathbf{Wp} + \mathbf{b})$

Mean Square Error: *(for ADALINE it is a quadratic fn.)*

$F(\mathbf{x}) = E[e^2] = E[(t-a)^2] = E[(t - \mathbf{x}^T\mathbf{z})^2]$

$F(\mathbf{x}) = c - 2\mathbf{x}^T\mathbf{h} + \mathbf{x}^T\mathbf{R}\mathbf{x}$,

$c = E[t^2]$, $\mathbf{h} = E[t\mathbf{z}]$ *and* $\mathbf{R} = E[\mathbf{z}\mathbf{z}^T] \Rightarrow A = 2\mathbf{R}, \mathbf{d} = -2\mathbf{h}$

Unique minimum, if it exists, is $\mathbf{x}^* = \mathbf{R}^{-1}\mathbf{h}$,

where $\mathbf{x} = \begin{bmatrix} {}_1\mathbf{w} \\ b \end{bmatrix}$ and $\mathbf{z} = \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix}$

LMS Algorithm: $\mathbf{W}(k+1) = \mathbf{W}(k) + 2\alpha\,\mathbf{e}(k)\,\mathbf{p}^T(k)$

$\mathbf{b}(k+1) = \mathbf{b}(k) + 2\alpha\,\mathbf{e}(k)$

Convergence Point: $\mathbf{x}^* = \mathbf{R}^{-1}\mathbf{h}$

Stable Learning Rate: $0 < \alpha < 1/\lambda_{max}$ where λ_{max} is the maximum eigenvalue of $\mathbf{R}$

Adaptive Filter ADALINE:

$$a(k) = purelin(\mathbf{Wp}(k) + b) = \sum_{i=1}^{R} w_{1,i}\,y(k-i+1) + b$$

Backpropagation Algorithm:

Performance Index:

Mean Square error: $F(\mathbf{x}) = E[\mathbf{e}^T\mathbf{e}] = E[(\mathbf{t}-\mathbf{a})^T(\mathbf{t}-\mathbf{a})]$

Approximate Performance Index: (single sample)

$\hat{F}(x) = \mathbf{e}^T(k)\mathbf{e}(k) = (\mathbf{t}(k) - \mathbf{a}(k))^T(\mathbf{t}(k) - \mathbf{a}(k))$

Sensitivity: $\mathbf{s}^m = \frac{\partial\hat{F}}{\partial\mathbf{n}^m} = \begin{bmatrix} \frac{\partial\hat{F}}{\partial n_1^m} & \frac{\partial\hat{F}}{\partial n_2^m} & \cdots & \frac{\partial\hat{F}}{\partial n_{S^m}^m} \end{bmatrix}^T$

Forward Propagation: $\mathbf{a}^0 = \mathbf{p}$,

$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^m + \mathbf{b}^{m+1})$ *for* $m = 0, 1, \ldots, M-1$

$\mathbf{a} = \mathbf{a}^M$

Backward Propagation: $\mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t} - \mathbf{a})$,

$\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T\mathbf{s}^{m+1}$ *for* $m = M-1, \ldots, 2, 1$,where

$\dot{\mathbf{F}}^m(\mathbf{n}^m) = \mathrm{diag}([\dot{f}^m(n_1^m) \;\; \dot{f}^m(n_2^m) \;\; \ldots \;\; \dot{f}^m(n_{S^m}^m)])$

$\dot{f}^m(n_j^m) = \frac{\partial f^m(n_j^m)}{\partial n_j^m}$

Weight Update (Approximate Steepest Descent):

$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha\mathbf{s}^m(\mathbf{a}^{m-1})^T$

$\mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha\mathbf{s}^m$

***Heuristic Variations of Backpropagation:**

Batching: The parameters are updated only after the entire training set has been presented. The gradients calculated for each training example are averaged together to produce a more accurate estimate of the gradient.(If the training set is complete, i.e., covers all possible input/output pairs, then the gradient estimate will be exact.)

Backpropagation with Momentum (MOBP):

$\Delta\mathbf{W}^m(k) = \gamma\Delta\mathbf{W}^m(k-1) - (1-\gamma)\alpha\,\mathbf{s}^m(\mathbf{a}^{m-1})^T$

$\Delta\mathbf{b}^m(k) = \gamma\Delta\mathbf{b}^m(k-1) - (1-\gamma)\alpha\,\mathbf{s}^m$

Variable Learning Rate Backpropagation (VLBP)

1. If the squared error (over the entire training set) increases by more than some set percentage ζ (typically one to five percent) after a weight update, then the weight update is discarded, the learning rate is multiplied by some factor $\rho < 1$, and the momentum coefficient γ (if it is used) is set to zero.
2. If the squared error decreases after a weight update, then the weight update is accepted and the learning rate is multiplied by some factor $\eta > 1$. If γ has been previously set to zero, it is reset to its original value.
3. If the squared error increases by less than ζ, then the weight update is accepted but the learning rate and the momentum coefficient are unchanged.

Association: $\mathbf{a} = hardlim(\mathbf{W}^0\mathbf{P}^0 + \mathbf{Wp} + b)$

An association is a link between the inputs and outputs of a network so that when a stimulus A is presented to the network, it will output a response B.

Associative Learning Rules:

Unsupervised Hebb Rule:

$\mathbf{W}(q) = \mathbf{W}(q-1) + \alpha\,\mathbf{a}(q)\mathbf{p}^T(q)$

Hebb with Decay:

$\mathbf{W}(q) = (1-\gamma)\mathbf{W}(q-1) + \alpha\,\mathbf{a}(q)\mathbf{p}^T(q)$

Instar: $a = hardlim(\mathbf{Wp} + b)$, $a = hardlim({}_1\mathbf{w}^T\mathbf{p} + b)$

The instar is activated for ${}_1\mathbf{w}^T\mathbf{p} = \|{}_1\mathbf{w}\|\|\mathbf{p}\|cos\theta \geq -b$

where θ is the angle between $\mathbf{p}$ and ${}_1\mathbf{w}$.

Instar Rule:

${}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha\,a_i(q)(\mathbf{p}(q) - {}_i\mathbf{w}(q-1))$

${}_i\mathbf{w}(q) = (1-\alpha)\;{}_i\mathbf{w}(q-1) + \alpha\,\mathbf{p}(q)$, *if* $(a_i(q) = 1)$

Kohonen Rule:

${}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha\,(\mathbf{p}(q) - {}_i\mathbf{w}(q-1))$ *for* $i \in X(q)$

Outstar Rule: $\mathbf{a} = satlins(\mathbf{Wp})$

$\mathbf{w}_j(q) = \mathbf{w}_j(q-1) + \alpha\,(\mathbf{a}(q) - \mathbf{w}_j(q-1))\,p_j(q)$

Competitive Layer: $\mathbf{a} = compet(\mathbf{Wp}) = compet(\mathbf{n})$

Competitive Learning with the Kohonen Rule:

${}_{i^*}\mathbf{w}(q) = {}_{i^*}\mathbf{w}(q-1) + \alpha\,(\mathbf{p}(q) - {}_{i^*}\mathbf{w}(q-1))$

$= (1-\alpha)\,{}_{i^*}\mathbf{w}(q-1) + \alpha\,\mathbf{p}(q)$

${}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1)$, $i \neq i^*$ where i^* is the winning neuron.

Self-Organizing with the Kohonen Rule:

${}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha\,(\mathbf{p}(q) - {}_i\mathbf{w}(q-1))$

$= (1-\alpha)\;{}_i\mathbf{w}(q-1) + \alpha\,\mathbf{p}(q)$, $i \in N_{i^*}(d)$

$N_i(d) = \{j, d_{i,j} \leq d\}$

LVQ Network: $(w_{k,i}^2 = 1) \Rightarrow$ subclass i is a part of class k

$n_i^1 = -\|{}_i\mathbf{w}^1 - \mathbf{p}\|$, $\mathbf{a}^1 = compet(\mathbf{n}^1)$, $\mathbf{a}^2 = \mathbf{W}^2\mathbf{a}^1$

LVQ Network Learning with the Kohonen Rule:

${}_{i^*}\mathbf{w}^1(q) = {}_{i^*}\mathbf{w}^1(q-1) + \alpha\,(\mathbf{p}(q) - {}_{i^*}\mathbf{w}^1(q-1))$,

if $a_{k^*}^2 = t_{k^*} = 1$

${}_{i^*}\mathbf{w}^1(q) = {}_{i^*}\mathbf{w}^1(q-1) - \alpha\,(\mathbf{p}(q) - {}_{i^*}\mathbf{w}^1(q-1))$,

if $a_{k^*}^2 = 1 \neq t_{k^*} = 0$

hardlim: $a = \begin{cases} 0 & n < 0 \\ 1 & n \geq 0 \end{cases}$, ***hardlims***: $a = \begin{cases} -1 & n < 0 \\ +1 & n \geq 0 \end{cases}$, ***purelin***: $a = n$, ***Logsig***: $a = \frac{1}{1+e^{-n}}$, ***tansig***: $a = \frac{e^n - e^{-n}}{e^n + e^{-n}}$, ***poslin***: $a = \begin{cases} 0 & n < 0 \\ n & n \geq 0 \end{cases}$

compet: $a = \begin{cases} 1 & \text{neuron with max } n \\ 0 & \text{all other neurons} \end{cases}$, ***satlin***: $a = \begin{cases} 0 & n < 0 \\ n & -1 \leq n \leq 1 \\ 1 & n > 1 \end{cases}$, ***satlins***: $a = \begin{cases} -1 & n < 0 \\ n & -1 \leq n \leq 1 \\ 1 & n > 1 \end{cases}$

Delay: $a(t) = u(t-1)$, ***Integrator***: $a(t) = \int_0^t u(\tau)d\tau + a(0)$

****HINT:**

$diag([1\;2\;3]) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$

The full list can be found here or here. It covers the following topics:

- Big-O Algorithm Cheat Sheet: http://bigocheatsheet.com/
- Bokeh Cheat Sheet: https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Python_Bo
- Data Science Cheat Sheet: https://www.datacamp.com/community/tutorials/python-data-science-ch...
- Data Wrangling Cheat Sheet: https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-c...
- Data Wrangling: https://en.wikipedia.org/wiki/Data_wrangling
- Ggplot Cheat Sheet: https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatshe...
- Keras Cheat Sheet: https://www.datacamp.com/community/blog/keras-cheat-

sheet#gs.DRKeNMs

- Keras: https://en.wikipedia.org/wiki/Keras
- Machine Learning Cheat Sheet: https://www.datasciencecentral.com/profiles/blogs/the-making-of-a-c...
- Machine Learning Cheat Sheet: https://docs.microsoft.com/en-in/azure/machine-learning/machine-lea...
- ML Cheat Sheet:: http://peekaboo-vision.blogspot.com/2013/01/machine-learning-cheat-...
- Matplotlib Cheat Sheet: https://www.datacamp.com/community/blog/python-matplotlib-cheat-she...
- Matpotlib: https://en.wikipedia.org/wiki/Matplotlib
- Neural Networks Cheat Sheet: http://www.asimovinstitute.org/neural-network-zoo/
- Neural Networks Graph Cheat Sheet: http://www.asimovinstitute.org/blog/
- Neural Networks: https://www.quora.com/Where-can-find-a-cheat-sheet-for-neural-network
- Numpy Cheat Sheet: https://www.datacamp.com/community/blog/python-numpy-cheat-sheet#gs...
- NumPy: https://en.wikipedia.org/wiki/NumPy
- Pandas Cheat Sheet: https://www.datacamp.com/community/blog/python-pandas-cheat-sheet#g...
- Pandas: https://en.wikipedia.org/wiki/Pandas_(software)
- Pandas Cheat Sheet: https://www.datacamp.com/community/blog/pandas-cheat-sheet-python#g...
- Pyspark Cheat Sheet: https://www.datacamp.com/community/blog/pyspark-cheat-sheet-python#...
- Scikit Cheat Sheet: https://www.datacamp.com/community/blog/scikit-learn-cheat-sheet
- Scikit-learn: https://en.wikipedia.org/wiki/Scikit-learn
- Scikit-learn Cheat Sheet: http://peekaboo-vision.blogspot.com/2013/01/machine-learning-cheat-...
- Scipy Cheat Sheet: https://www.datacamp.com/community/blog/python-scipy-cheat-sheet#gs...
- SciPy: https://en.wikipedia.org/wiki/SciPy
- TesorFlow Cheat Sheet: https://www.altoros.com/tensorflow-cheat-sheet.html

To not miss this type of content in the future, subscribe to our newsletter. Other articles you might be interested in:

- Book and Resources for DSC Members
- Comprehensive Repository of Data Science and ML Resources
- Advanced Machine Learning with Basic Excel
- Difference between ML, Data Science, AI, Deep Learning, and Statistics
- Selected Business Analytics, Data Science and ML articles
- Hire a Data Scientist | Search DSC | Find a Job
- Post a Blog | Forum Questions

Viewed using Just Read