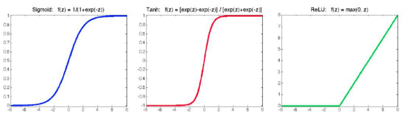
Activation Functions



$\sigma(z)=\frac{1}{1+\exp(-z)}$ $\tanh(z)=\frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$ $\mathrm{ReLU}(z)=\max(0,z)$

$\sigma'(z)=\sigma(z)*(1-\sigma(z))$ $\tanh'(z)=1-\tanh(z)^2$

有可能梯度消失 有可能梯度消失 不可能梯度消失 0处无导数(梯度)

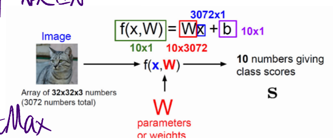
Single Layer Neural Network

$\mathbf{s}=\mathbf{W}^T\mathbf{x}$ vector of sums from weight matrix $\mathbf{h}=\sigma(\mathbf{s})$

Winner-Take-All (WTA)

$$h_j=g(\mathbf{s})=\begin{cases}1 & \text{if } j=\arg\max_i \mathbf{w}_i^T\mathbf{x}\\ 0 & \text{if otherwise}\end{cases}$$

E.g 分类任务



SoftMax

$L_i=-\log(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}})$

unnormalized probabilities

	unnormalized log probabilities	exp		normalize	probabilities	
cat	3.2	→	24.5	→	0.13	→ L_i = -log(0.13) = 0.89
car	5.1		164.0		0.87	
frog	-1.7		0.18		0.00	

正则

In common use:

L2 regularization $R(W)=\sum_k\sum_l W_{k,l}^2$ 防止过分依赖某个权重

L1 regularization $R(W)=\sum_k\sum_l |W_{k,l}|$ 非重要权重归0

Elastic net (L1 + L2) $R(W)=\sum_k\sum_l \beta W_{k,l}^2+|W_{k,l}|$

Supervised learning framework

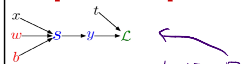
$$\mathbf{w}^*=\underset{\mathbf{w}}{\operatorname{argmin}}\sum_{n=1}^{N}\operatorname{loss}(\mathbf{o}^{(n)},\mathbf{t}^{(n)})$$ o: output t: target

Squared loss: $\sum_k\frac{1}{2}(o_k^{(n)}-t_k^{(n)})^2$

Cross-entropy loss: $-\sum_k t_k^{(n)}\log o_k^{(n)}$

$\mathbf{w}^{t+1}=\mathbf{w}^t-\eta\frac{\partial E}{\partial \mathbf{w}^t}$

Computation Graph



$s=wx+b$

$y=\sigma(s)$

$\mathcal{L}=\frac{1}{2}(y-t)^2$

1. 根据已知列式,将结果列图:

2. 写Chain Rule: $\frac{\partial L}{\partial b}=\frac{\partial L}{\partial y}\cdot\frac{\partial y}{\partial s}\cdot\frac{\partial s}{\partial b}$

3. 分别求导后代入 $\frac{\partial L}{\partial b}=(y-t)\cdot\sigma(s)\cdot(1-\sigma(s))\cdot 1$ ✓

Stochastic Gradient Descent: compute error using a single sample at a time, update weights, repeat

Batch Gradient Descent: compute error on all examples, update weights based on error, repeat

★ **Mini-batch Gradient Descent**: randomly select a subset from the training data, calculate error on subset, update weights, repeat

Principal Component Analysis (PCA)

What is PCA: Unsupervised technique for extracting variance structure from high dimensional datasets.

PCA is an orthogonal projection or transformation of the data into a (possibly lower dimensional) subspace so that the variance of the projected data is maximized.

Find vector that maximizes sample variance of projected data

$$\frac{1}{n}\sum_{i=1}^{n}(\mathbf{v}^T\mathbf{x}_i)^2=\mathbf{v}^T\mathbf{X}\mathbf{X}^T\mathbf{v}\qquad \max_{\mathbf{v}}\ \mathbf{v}^T\mathbf{X}\mathbf{X}^T\mathbf{v}\quad \text{s.t.}\quad \mathbf{v}^T\mathbf{v}=1$$

$$\max_{\mathbf{v}}\mathbf{v}^T\mathbf{X}\mathbf{X}^T\mathbf{v}-\lambda\mathbf{v}^T\mathbf{v}\qquad \partial/\partial\mathbf{v}=0\qquad (\mathbf{X}\mathbf{X}^T-\lambda\mathbf{I})\mathbf{v}=0$$

$$\Rightarrow (\mathbf{X}\mathbf{X}^T)\mathbf{v}=\lambda\mathbf{v}$$

$(XX^T)v=\lambda v$, so v (the first PC) is the eigenvector of sample correlation/covariance matrix XX^T

Original representation	Transformed representation
Data point $x_i=(x_i^1,\ldots,x_i^D)$ ⇒	projection $(v_1\cdot x^i,\ldots,v_d\cdot x^i)$
D-dimensional vector	d-dimensional vector

Strengths	**Weaknesses**
Eigenvector method	Limited to second order statistics
No tuning of the parameters	Limited to linear projections
No local optima	

PCA计算:

① 计算均值,减去得到去中心化的值

② 计算协方差矩阵C: $\begin{bmatrix}\mathrm{cov}(x,x) & \mathrm{cov}(x,y)\\ \mathrm{cov}(y,x) & \mathrm{cov}(y,y)\end{bmatrix}$

$$\mathrm{cov}(x,y)=\frac{\sum(x_i-x_{mean})(y_i-y_{mean})}{n-1}$$

(若去中心,直接为 x_iy_i,n-1不除无偏估

③ 解特征值,

$\det(C-\lambda I)=0\Rightarrow\det\begin{bmatrix}\mathrm{cov}(x,x)-\lambda & \mathrm{cov}(x,y)\\ \mathrm{cov}(y,x) & \mathrm{cov}(y,y)-\lambda\end{bmatrix}=0$

解出λ ⇒ $(C-\lambda_1 I)v_1=0$ 解出V为主成分向量,

λ为方差.

Kernel PCA

Key Idea: Replace inner product matrix by kernel matrix

PCA: $\frac{1}{n}X^TXX^TX\alpha=\lambda X^TX\alpha$

Kernel PCA: replace "X^TX" with K.

$\frac{1}{n}KK\alpha=\lambda K\alpha$, or equivalently, $\frac{1}{n}K\alpha=\lambda\alpha$

Key computation: form an n by n kernel matrix K, and then perform eigen-decomposition on K.

作用: 解决非线性问题,先映射到高维再PCA分解.

步骤: 1. 中心化(减均值)

2. 定义核K,大小为n×n, $K(i,j)=e^{(-\gamma\|x(i)-x(j)\|^2)}$

γ为高斯的核参数

3. 特征值分解,选择主成分

4. 投影到新空间 K(i)通过V映射

5. 线性分割(如SVM)

k-Means Clustering Algorithm

Problem formulation:

- Given a sample $\mathcal{X}=\{\mathbf{x}^t\}_{t=1}^N$.
- Find k reference vectors (or prototypes or codebook vectors or codewords) $\mathbf{m}_j$ which best represent the data.

Encoding: from a data point $\mathbf{x}^t$ to the index i of a reference vector.
Decoding: from an index i to the corresponding reference vector $\mathbf{m}_i$.

Initialize $\mathbf{m}_i$, $i=1,\ldots,k$ (e.g., k randomly selected $\mathbf{x}^t$)
Repeat
For all $\mathbf{x}^t\in\mathcal{X}$, we obtain the estimated labels

$$b_i^t=\begin{cases}1 & \text{if } i=\arg\min_j\|\mathbf{x}^t-\mathbf{m}_j\|\\ 0 & \text{otherwise}\end{cases}$$

For all $\mathbf{m}_i$, $i=1,\ldots,k$, we obtain (by taking the derivative of $E(\{\mathbf{m}_i\}_{i=1}^k\mid\mathcal{X})$ with respect to $\mathbf{m}_i$ and setting it to $\mathbf{0}$)

$$\mathbf{m}_i=\frac{\sum_t b_i^t\mathbf{x}^t}{\sum_t b_i^t}$$

The reference vector is set to the mean (center) of all the instances that it represents.
Until $\mathbf{m}_i$ converge.

- Each data point $\mathbf{x}^t$ is represented by the index i of the nearest reference vector:
$$i=\arg\min_j\|\mathbf{x}^t-\mathbf{m}_j\|$$
and we can compute the labels $\mathbf{b}^t$ for $\mathbf{x}^t$ as
$$b_i^t=\begin{cases}1 & \text{if } i=\arg\min_j\|\mathbf{x}^t-\mathbf{m}_j\|\\ 0 & \text{otherwise}\end{cases}$$
- Encoding can lead to data compression: instead of storing (or transmitting) $\mathbf{x}^t$, we only need to store (or transmit) i.
- Since $\mathbf{x}^t$ is represented by $\mathbf{m}_i$ after encoding and then decoding, reconstruction error $\|\mathbf{x}^t-\mathbf{m}_i\|^2$ is incurred. The total reconstruction error is
$$E(\{\mathbf{m}_i\}_{i=1}^k\mid\mathcal{X})=\sum_t\sum_i b_i^t\|\mathbf{x}^t-\mathbf{m}_i\|^2$$

Lagrangian

minimize$_{\mathbf{x}}$ $f_0(\mathbf{x})$

subject to $f_i(\mathbf{x})\le 0$, $i=1,\ldots,m$

(assume $\mathbf{x}\in\mathbb{R}^n$) Lagrangian $\mathcal{L}:\mathbb{R}^{n+m}\to\mathbb{R}$

$$\mathcal{L}(\mathbf{x},\boldsymbol{\lambda})=f_0(\mathbf{x})+\lambda_1f_1(\mathbf{x})+\ldots+\lambda_mf_m(\mathbf{x})=f_0(\mathbf{x})+\sum_i\lambda_if_i(\mathbf{x})$$

- λ_i: Lagrange multipliers or dual variables, which can be considered as "costs" of violating the corresponding constraints
- objective is augmented with weighted sum of constraint functions

Example: linear programming (LP) (inequality form)

minimize$_{\mathbf{x}}$ $\mathbf{c}^T\mathbf{x}$ subject to $\mathbf{a}_i^T\mathbf{x}-b_i\le 0$, $i=1,\ldots,m$

$$\mathcal{L}(\mathbf{x},\boldsymbol{\lambda})=\mathbf{c}^T\mathbf{x}+\sum_{i=1}^m\lambda_i(\mathbf{a}_i^T\mathbf{x}-b_i)=-\mathbf{b}^T\boldsymbol{\lambda}+(\mathbf{A}\boldsymbol{\lambda}+\mathbf{c})^T\mathbf{x}$$

?

$$g(\boldsymbol{\lambda})=\begin{cases}-\mathbf{b}^T\boldsymbol{\lambda} & \text{if } \mathbf{A}\boldsymbol{\lambda}+\mathbf{c}=\mathbf{0}\\ -\infty & \text{otherwise}\end{cases}$$

Primal: minimize$_{\mathbf{x}}$ $\mathbf{c}^T\mathbf{x}$ subject to $\mathbf{A}\mathbf{x}\le\mathbf{b}$

dual of LP is also an LP

Dual: maximize$_{\boldsymbol{\lambda}}$ $-\mathbf{b}^T\boldsymbol{\lambda}$ subject to $\mathbf{A}^T\boldsymbol{\lambda}+\mathbf{c}=\mathbf{0}$, $\boldsymbol{\lambda}>\mathbf{0}$

KKT Optimality Conditions

suppose

- f_0 and f_i are differentiable
- $\mathbf{x}^*,\boldsymbol{\lambda}^*$ are (primal, dual) optimal, with zero duality gap

minimize$_{\mathbf{x}}$ $f_0(\mathbf{x})$

subject to $f_i(\mathbf{x})\le 0$, $i=1,\ldots,m$

$h_i(\mathbf{x})=0$, $i=1,\ldots,p$

define Lagrangian $\mathcal{L}:\mathbb{R}^{n+m+p}\to\mathbb{R}$ as

$$\mathcal{L}(\mathbf{x},\boldsymbol{\lambda},\boldsymbol{\nu})=f_0(\mathbf{x})+\sum_{i=1}^m\lambda_if_i(\mathbf{x})+\sum_{i=1}^p\nu_ih_i(\mathbf{x})$$

dual function: $g(\boldsymbol{\lambda},\boldsymbol{\nu})=\inf_{\mathbf{x}}\mathcal{L}(\mathbf{x},\boldsymbol{\lambda},\boldsymbol{\nu})$

- $(\boldsymbol{\lambda},\boldsymbol{\nu})$ is dual feasible if $\boldsymbol{\lambda}\ge 0$ and $g(\boldsymbol{\lambda},\boldsymbol{\nu})>-\infty$

assume f_i,h_i differentiable
if $\mathbf{x}^*,\boldsymbol{\lambda}^*,\boldsymbol{\nu}^*$ are optimal, with zero duality gap, then they satisfy KKT conditi

$f_i(\mathbf{x}^*)\le 0$, $h_i(\mathbf{x}^*)=0$ (primal feasibility)

$\lambda_i^*\ge 0$ (dual feasibility)

$\lambda_i^*f_i(\mathbf{x}^*)=0$ (complementary)

$\nabla f_0(\mathbf{x}^*)+\sum_i\lambda_i^*\nabla f_i(\mathbf{x}^*)+\sum_i\nu_i^*\nabla h_i(\mathbf{x}^*)=0$ (stationarity)

KKT的疑惑

$f(x,y) = x^2+y^2$

$g_1(y) = y <= a$

$L(x,y,\lambda_1,\lambda_2) = x+y^2+\lambda(y-a)$

KKT条件: $\frac{\partial L}{\partial x} = 2x = 0$ $\frac{\partial L}{\partial y} = 2y+\lambda = 0$

$y \le a$ $\lambda \ge 0$ $\lambda(y-a) = 0$

$\Rightarrow x = 0$ $y = -\frac{\lambda}{2}$ $y \le a$ $\lambda \ge 0$ $\lambda(y-a) = 0$

① $\lambda = 0$, $x = 0$, $y = 0$ $a \ge y \ge 0$

② $\lambda = -2a$, $x = 0$, $y = a$ $\lambda \ge 0 \Rightarrow a \le 0$

若 $a \ge 0$ 解为 $(0,0)$ 若 $a < 0$, 解为 $(0,a)$

CNN参数计算:

Input: W×H×D Kernel: F×F×D, 数量为K

Padding = P Stride = S 求Output

$W_0 = \frac{W-F+2P}{S}+1$ $H_0 = \frac{H-F+2P}{S}+1$

Output: $W_0 \times H_0 \times K$

Params = $(F \times F \times D) \times K$ $(+k)$ if Bias

Kernel Flipping 水平垂直翻转: $\begin{bmatrix} a & b \\ c & d \end{bmatrix} \rightarrow \begin{bmatrix} d & c \\ b & a \end{bmatrix}$

全连接参数: Hin×Hout +Hout if Bias

Deep Belief Networks I

- Problems with back-propagation for MLP training:
  - It requires labeled training data. (Almost all data is unlabeled.)
  - The learning time does not scale well. (It is very slow in networks with multiple hidden layers.)
  - It can get stuck in poor local optima.

The hidden states can be inferred in an efficient bottom-up fashion.

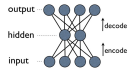
Training of the RBMs is done in a greedy layer-wise manner starting from the bottommost layer.

It can be shown that this inference procedure increases a lower bound of the observed data likelihood.

After the greedy layer-wise training procedure, it is common to fine-tune the weights using a technique called backfitting which involves both bottom-up and top-down operations. However, this procedure is very slow.

Autoencoders I

- An autoencoder (AE) is a feedforward neural network for learning a compressed representation or latent representation (encoding) of the input data by learning to predict the input itself in the output.



- The hidden layer in the middle is constrained to be a narrow bottleneck (with fewer units than the input and output layers) to ensure that the hidden units capture the most relevant aspects of the data.
- The input-to-hidden part corresponds to an encoder while the hidden-to-output part corresponds to a decoder.

Minimize the reconstruction loss

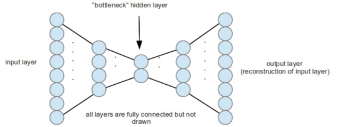
$$\underset{\mathbf{W}_1,\mathbf{W}_2,\mathbf{w}_{01},\mathbf{w}_{02}}{\text{minimize}} \frac{1}{2}\sum_{i=1}^{N}\left\|\mathbf{x}^{(i)} - \underbrace{(\mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{x}^{(i)}+\mathbf{w}_{10})+\mathbf{w}_{20})}_{=\hat{\mathbf{x}}^{(i)}}\right\|_2^2$$

For linear hidden units, it can be shown that the weights to the k hidden units span the same subspace as the first k principal components of the data, i.e., linear autoencoders are equivalent to principal component analysis (PCA) (to be covered in Dimensionality Reduction topic).

Nonlinear representations of the data can be obtained by using nonlinear hidden units, e.g., units with the sigmoid function.

A deep autoencoder (or stacked autoencoder) is a multilayer extension of the simple autoencoder by using multiple hidden layers.

The middle layer still acts as a bottleneck.



Training deep autoencoders using the standard backpropgation learning algorithm does not work well because:

- the gradient signal becomes too small as it passes back through multiple layers
- the learning algorithm often gets stuck in poor local minima.

One solution is to train a series of RBMs through a pretraining procedure and then use them to initialize an autoencoder. Afterwards, the whole network is fine-tuned using backpropgation in the usual way.

Low-Level to High-Level Features

Convolutional Neural Networks

The development of convolutional neural networks (CNN), or convolutional networks (ConvNet), has been inspired by neurophysiological experiments that David H. Hubel and Torsten N. Wiesel (winners of the Nobel Prize in Physiology or Medicine 1981) conducted in the 50s and 60s to understand how the mammalian vision system works.

Some neurons in the visual cortex will be associated with specific local receptive fields.

CNNs are a special type of feedforward neural networks for processing data with a known grid-like topology, e.g.

- Spatial data: 2 spatial dimensions
- Spatiotemporal data: 1 spatial dimension, 1 temporal dimension

At least one layer (called convolutional layer) in a CNN uses convolution in place of matrix multiplication.

Finite 2-D domains:

$$s[i,j] = (x*w)[i,j] = \sum_{m=-M}^{M}\sum_{n=-N}^{N} x[m,n]w[i-m,j-n]$$
$$= \sum_{m=-M}^{M}\sum_{n=-N}^{N} x[i-m,j-n]w[m,n]$$

Sparse connectivity:

- The kernel is (much) smaller than the input.
- Sparse weights in CNNs vs. fully-connected weights in conventional neural networks:

Parameter sharing:

- The parameters of a kernel are the same when applied to different locations of the input.

Pooling

A pooling function replaces the output at a certain location with a summary statistic of the nearby outputs.

It helps to make the higher-level representation invariant to small changes of the input and hence more robust.

Common pooling operations:

- Max pooling: reports the maximum output within a rectangular neighborhood.
- Average pooling: reports the average output of a rectangular neighborhood (possibly weighted by the distance from the central pixel).

By spacing pooling regions $k > 1$ (rather than 1) pixels apart, the next higher layer has roughly k times fewer inputs to process, leading to downsampling.

Dropout

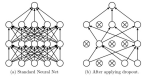
Deep models have many learnable parameters and hence are prone to overfitting.

To prevent overfitting, one way is to use a very large dataset but this is not desirable for supervised learning methods (e.g., CNN) which need labeled training data.

A powerful alternative is the method called dropout which may be seen as a computationally very cheap realization (or approximation) of an ensemble method called bagging (to be covered later).

Dropout in CNNs

Dropout is very powerful because it effectively trains and evaluates a bagged ensemble of exponentially many deep models which are sub-networks formed by removing (randomly with a dropout rate of say 0.5) units or weights (achieved simply via multiplying their values by zero) from the original CNN.
Dropout is applied to the fully-connected layers of a CNN.



Dropout is actually a general regularization technique which can also be applied to other models that use a distributed representation and can be trained with stochastic gradient descent, e.g., feedforward neural networks, recurrent neural networks, and probabilistic models such as RBM.

Introduction to Recurrent Neural Networks

Recurrent neural networks (RNNs) are extensions of feedforward neural networks for handling sequential data (such as sound, time series (sensor) data, or written natural language) typically involving variable-length input or output sequences.

There is weight sharing in an RNN such that the weights are shared across different instances of the units corresponding to different time steps.

Weight sharing is important for processing variable-length sequences so that the absolute time step at which an event occurs does not matter but the context (relative to some other events) in which an event occurs is more important and relevant.

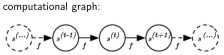
The computation involved in a dynamical system defined recursively can be unfolded to form a computational graph with a repetitive structure such as a chain of events.

A classical dynamical system:

$$\mathbf{s}^{(t)} = f(\mathbf{s}^{(t-1)};\boldsymbol{\theta})$$

where $\mathbf{s}^{(t)}$ denotes the state of the system at time t and $\boldsymbol{\theta}$ denotes the system parameters which are time-invariant (effectively achieving weight sharing).

Corresponding computational graph:



Computations occurring at different time steps of the recurrent structure (where the black square indicates a delay of 1 time step) are represented as different nodes in the computational graph.

the hidden state $\mathbf{h}^{(t)}$ may be thought of as a kind of (lossy) summary of the past input sequences up to time t, i.e., $\mathbf{x}^{(\tau)}$ for $\tau \le t-1$.

Deep RNNs

Many ways to increase the network depth, e.g.,

- Multiple recurrent hidden layers
- Deeper computation in recurrent layers
- Mixture of different recurrent connection types (path-lengthening effect may be mitigated by introducing skip connections)

Recursive neural networks generalize ordinary RNNs in that the unfolded computational graphs are no longer chains but trees.

Recursive neural networks can take data structures that are more general than feature vectors as input.

They have been used successfully for natural language processing and computer vision applications. (You can take a course on "Deep Learning for Natural Language Processing".)

To process a sequence of length N, an RNN has a length of N but a recursive neural network reduces it to $O(\log N)$. This helps the latter to deal with long-term dependencies more effectively.

Learning long-term dependencies in RNNs is challenging because:

- the gradients propagated over many stages tend to vanish (most of the time) or explode (sometimes)
- exponentially smaller weights are given to long-term interactions involving the multiplication of many Jacobians.

The use of long short-term memory (LSTM) cells is one way proposed to help the learning of long-term dependencies in RNNs.

Long Short-Term Memory

LSTM is based on the idea of leaky units which allow an RNN to accumulate information over a longer duration (e.g., we use a leaky unit to accumulate evidence for each subsequence inside a sequence).

Once the information accumulated is used (e.g., suffcient evidence has been accumulated for a subsequence), we need a mechanism to forget the old state by setting it to zero and starting to count again from scratch.

Instead of manually deciding when to clear the state, LSTM provides a way to learn to decide when to do it by conditioning the forgetting on the context.

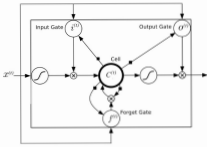
Gradients can flow for long durations through a linear self-loop whose weight is controlled by another hidden unit to change the time scale of the integration dynamically.

LSTM Cell

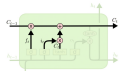
An LSTM recurrent neural network consists of LSTM cells as hidden units which have an internal recurrence in addition to the recurrence in the recurrent network. LSTM RNN is a kind of gated RNN.

Each LSTM cell has the same inputs and outputs as an ordinary RNN cell, but has more parameters and a system of gating units to control the information flow.

3 gates (or gate controllers) in an LSTM cell: external input gate, forget gate, output gate

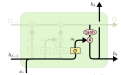


Cell state:



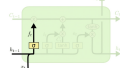
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Output gate:



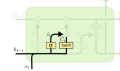
$$o_t = \sigma(W_o\,[h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * \tanh(C_t)$$

Forget gate:



$$f_t = \sigma(W_f\cdot[h_{t-1}, x_t] + b_f)$$

External Input gate:



$$i_t = \sigma(W_i\cdot[h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh(W_C\cdot[h_{t-1}, x_t] + b_C)$$

$$
\begin{aligned}
(\mathbf{AB})^{-1} &= \mathbf{B}^{-1}\mathbf{A}^{-1} \\
\mathbf{ABC...})^{-1} &= ...\mathbf{C}^{-1}\mathbf{B}^{-1}\mathbf{A}^{-1} \\
(\mathbf{A}^T)^{-1} &= (\mathbf{A}^{-1})^T \\
(\mathbf{A}+\mathbf{B})^T &= \mathbf{A}^T + \mathbf{B}^T \\
(\mathbf{AB})^T &= \mathbf{B}^T\mathbf{A}^T \\
(\mathbf{ABC...})^T &= ...\mathbf{C}^T\mathbf{B}^T\mathbf{A}^T \\
(\mathbf{A}^H)^{-1} &= (\mathbf{A}^{-1})^H \\
(\mathbf{A}+\mathbf{B})^H &= \mathbf{A}^H + \mathbf{B}^H \\
(\mathbf{AB})^H &= \mathbf{B}^H\mathbf{A}^H \\
(\mathbf{ABC...})^H &= ...\mathbf{C}^H\mathbf{B}^H\mathbf{A}^H
\end{aligned}
$$

$$
\begin{aligned}
\partial\mathbf{A} &= 0 \\
\partial(\alpha\mathbf{X}) &= \alpha\partial\mathbf{X} \\
\partial(\mathbf{X}+\mathbf{Y}) &= \partial\mathbf{X} + \partial\mathbf{Y} \\
\partial(\mathrm{Tr}(\mathbf{X})) &= \mathrm{Tr}(\partial\mathbf{X}) \\
\partial(\mathbf{XY}) &= (\partial\mathbf{X})\mathbf{Y} + \mathbf{X}(\partial\mathbf{Y}) \\
\partial(\mathbf{X}\circ\mathbf{Y}) &= (\partial\mathbf{X})\circ\mathbf{Y} + \mathbf{X}\circ(\partial\mathbf{Y}) \\
\partial(\mathbf{X}\otimes\mathbf{Y}) &= (\partial\mathbf{X})\otimes\mathbf{Y} + \mathbf{X}\otimes(\partial\mathbf{Y}) \\
\partial(\mathbf{X}^{-1}) &= -\mathbf{X}^{-1}(\partial\mathbf{X})\mathbf{X}^{-1} \\
\partial(\det(\mathbf{X})) &= \mathrm{Tr}(\mathrm{adj}(\mathbf{X})\partial\mathbf{X}) \\
\partial(\det(\mathbf{X})) &= \det(\mathbf{X})\mathrm{Tr}(\mathbf{X}^{-1}\partial\mathbf{X}) \\
(\ln(\det(\mathbf{X}))) &= \mathrm{Tr}(\mathbf{X}^{-1}\partial\mathbf{X}) \\
\partial\mathbf{X}^T &= (\partial\mathbf{X})^T \\
\partial\mathbf{X}^H &= (\partial\mathbf{X})^H
\end{aligned}
$$

$$
\frac{\partial\mathbf{x}^T\mathbf{a}}{\partial\mathbf{x}} = \frac{\partial\mathbf{a}^T\mathbf{x}}{\partial\mathbf{x}} = \mathbf{a}
$$

$$
\frac{\partial\mathbf{a}^T\mathbf{X}\mathbf{b}}{\partial\mathbf{X}} = \mathbf{a}\mathbf{b}^T
$$

$$
\frac{\partial\mathbf{a}^T\mathbf{X}^T\mathbf{b}}{\partial\mathbf{X}} = \mathbf{b}\mathbf{a}^T
$$

$$
\frac{\partial\mathbf{a}^T\mathbf{X}\mathbf{a}}{\partial\mathbf{X}} = \frac{\partial\mathbf{a}^T\mathbf{X}^T\mathbf{a}}{\partial\mathbf{X}} = \mathbf{a}\mathbf{a}^T
$$

$$
\frac{\partial\mathbf{X}}{\partial X_{ij}} = \mathbf{J}^{ij}
$$

$$
\frac{\partial(\mathbf{XA})_{ij}}{\partial X_{mn}} = \delta_{im}(\mathbf{A})_{nj} = (\mathbf{J}^{mn}\mathbf{A})_{ij}
$$

$$
\frac{\partial(\mathbf{X}^T\mathbf{A})_{ij}}{\partial X_{mn}} = \delta_{in}(\mathbf{A})_{mj} = (\mathbf{J}^{nm}\mathbf{A})_{ij}
$$

$$
\frac{\partial}{\partial X_{ij}}\sum_{klmn} X_{kl}X_{mn} = 2\sum_{kl} X_{kl}
$$

$$
\frac{\partial\mathbf{b}^T\mathbf{X}^T\mathbf{X}\mathbf{c}}{\partial\mathbf{X}} = \mathbf{X}(\mathbf{b}\mathbf{c}^T + \mathbf{c}\mathbf{b}^T)
$$

$$
\frac{\partial(\mathbf{Bx}+\mathbf{b})^T\mathbf{C}(\mathbf{Dx}+\mathbf{d})}{\partial\mathbf{x}} = \mathbf{B}^T\mathbf{C}(\mathbf{Dx}+\mathbf{d}) + \mathbf{D}^T\mathbf{C}^T(\mathbf{Bx}+\mathbf{b})
$$

$$
\frac{\partial(\mathbf{X}^T\mathbf{BX})_{kl}}{\partial X_{ij}} = \delta_{lj}(\mathbf{X}^T\mathbf{B})_{ki} + \delta_{kj}(\mathbf{BX})_{il}
$$

$$
\frac{\partial(\mathbf{X}^T\mathbf{BX})}{\partial X_{ij}} = \mathbf{X}^T\mathbf{B}\mathbf{J}^{ij} + \mathbf{J}^{ji}\mathbf{B}\mathbf{X} \qquad (\mathbf{J}^{ij})_{kl} = \delta_{ik}\delta_{jl}
$$

$$
\frac{\partial\mathbf{x}^T\mathbf{B}\mathbf{x}}{\partial\mathbf{x}} = (\mathbf{B}+\mathbf{B}^T)\mathbf{x}
$$

$$
\frac{\partial\mathbf{b}^T\mathbf{X}^T\mathbf{D}\mathbf{X}\mathbf{c}}{\partial\mathbf{X}} = \mathbf{D}^T\mathbf{X}\mathbf{b}\mathbf{c}^T + \mathbf{D}\mathbf{X}\mathbf{c}\mathbf{b}^T
$$

$$
\frac{\partial}{\partial\mathbf{X}}(\mathbf{Xb}+\mathbf{c})^T\mathbf{D}(\mathbf{Xb}+\mathbf{c}) = (\mathbf{D}+\mathbf{D}^T)(\mathbf{Xb}+\mathbf{c})\mathbf{b}^T
$$