Neuronal Networks
Formulae Collection

Marvin Ritter
marvin.ritter@gmail.com

14th September 2015

Introduction

This is my personal collection of formulae in the field of neural networks. I started it in preparation to my exam on the neural nets at Karlsruhe Institute of Technology (KIT). Even so the content will be similar to the course it is neither limited to it nor linked with the course in any means. *This is not official script and may contain errors and lag completeness.*

Corrections, supplements (or wishes) and links to good sources/ papers are very welcome. Just mail me to marvin.ritter@gmail.com.

Contents

Chapter 1

Vector Quantization

TODO

- N observations/training examples, $x_1, ..., x_N$
- fixed number of clusters/classes K
- $C(i) = K$ assigns observation x_i to a cluster k (typical each observation is only in one cluster)
- μ_k: centroid of cluster k
- N_k: number of observations that belong to cluster k ($N_k = \sum\limits_{C(i)=k} 1$)

1.1 Unsupervised Vector Quantization

Also called clustering.

1.1.1 Loss Functions

- Intra-class scatter: $W(C) = \frac{1}{2} \sum\limits_{k=1}^{K} \sum\limits_{C(i)=k} \sum\limits_{C(j)=k} d(x_i, x_j)$
- Inter-class scatter: $B(C) = \frac{1}{2} \sum\limits_{k=1}^{K} \sum\limits_{C(i)=k} \sum\limits_{C(j)\neq k} d(x_i, x_j)$
- Total scatter: $T(C) = W(C) + B(C) = \frac{1}{2} \sum\limits_{i=1}^{N} \sum_{j=1}^{N} d(x_i, x_j)$
- Minimizing $W(C)$ is equivalent to maximizing $B(C)$

1.1.2 k-Means

Minimize

$$W(C) = \frac{1}{2} \sum_{k=1}^{K} \sum_{C(i)=k} \sum_{C(j)=k} ||x_i - x_j||^2 = \sum_{k=1}^{K} N_k \sum_{C(i)=k} ||x_i - \mu_k||^2$$

1.1.3 Lloyd's Algorithm

1. **Classify**: Assign each observation i to the nearest centroid:

$$C(i) = \underset{1 \leq k \leq K}{\arg\min} \, ||x_i - \mu_k||^2$$

2. **Recenter**: For each class k, compute a new centroid as the mean of the updated class assignments:

$$\mu_k = \frac{\sum\limits_{C(i)=k} x_i}{N_k}$$

3. **Repeat until stopping criteria fulfilled**

1.2 Supervised Vector Quantization

1.2.1 Distortion Measures

TODO

Most common is the squared-error distortion:

$$d(x, \hat{x}) = \sum_{i=1}^{N} |x_i - \hat{x}_i|^2 \tag{1.1}$$

Other common distortion measures are the l_ν, or Holder norm,

$$d(x, \hat{x}) = \left(\sum_{i=1}^{N} |x_i - \hat{x}_i|^\nu \right)^{\frac{1}{\nu}} = ||x - \hat{x}||_\nu \tag{1.2}$$

and its ν^{th} power, the ν^{th}-law distortion:

$$d(x, \hat{x}) = \sum_{i=1}^{N} |x_i - \hat{x}_i|^\nu \tag{1.3}$$

The holder Norm (1.2) is a distance and fulfills the triangle inequality[1], the ν^{th}-law distortion not.

All three and many others, as the weighted-squares distortion and the quadratic distortion, depend on the difference $x - \hat{x}$. We call them can be described as $d(x, \hat{x}) = L(x - \hat{x})$. A distortion not having this form is the one by Itakura, Saito and Chaffee,

$$d(x, \hat{x}) = (x - \hat{x}) \, R(x) \, (x - \hat{x})^T \tag{1.4}$$

, where $R(x)$ is the autocorrelation matrix, see [Linde et al., 1980] for details.

[1] triangle inequality: $d(x, \hat{x}) \leq d(x, y) + d(y, \hat{x})$, for all y

1.2.2 Learning Vector Quantization

1.2.3 LVQ 1

1.2.4 LVQ 2.1

1.2.5 LVQ 3

Chapter 2

Multi-Layer Perceptrons

2.1 Backpropagation

- m inputs/features, $x \in \mathbb{R}^m$
- k target outputs, $t \in \mathbb{R}^k$
- n training examples of form $(x, t) \in \mathbb{R}^m \times \mathbb{R}^k$
- L layers $(1, \ldots, L)$
- E: error function (e. g. $E_{\text{MSE}} = \frac{1}{2} \sum\limits_{i=1}^{k} (t_i - o_i^{(L)})^2$)
- $\phi(y)$: activation function (e. g. $\phi(y) = \sigma(y) = \frac{1}{1+e^{-y}}$)
- $x_{ij}^{(l)}$: input i of neuron j in layer l
- $w_{ij}^{(l)}$: weight of connection from neuron i in layer $l-1$ to neuron j in layer l
- $z_j^{(l)} = \sum\limits_i w_{ij}^{(l)} * x_i^{(l-1)}$
- $a_j^{(l)} = \phi(z_j^{(l)})$
- $o = a^{(L)}$ is the output of the neural network
- η: learning rate (usually $\eta < 1$)
- $b^{(l)}$: bias unit layer l, either 1 if there is a weight for it, or without weight if $b^{(l)}$ is adjusted directly during training

$$
\begin{aligned}
\frac{\partial E}{\partial w_{ij}^{(L)}} &= \frac{\partial E}{\partial z_j^{(L)}} \frac{\partial z_j^{(L)}}{\partial w_{ij}^{(L)}} = \frac{\partial E}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial z_j^{(L)}}{\partial w_{ij}^{(L)}} \\
\frac{\partial z_j^{(L)}}{\partial w_{ij}^{(L)}} &= \frac{\partial \sum_i w_{ij}^{(L)} * x_i^{(L-1)}}{\partial w_{ij}^{(L)}} = x_i^{(L-1)} \\
\frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} &= \frac{\partial \phi(z_j^{(L)})}{\partial z_j^{(L)}} = (\phi(z_j^{(L)})(1-\phi(z_j^{(L)})) = a_j^{(L)}(1-a_j^{(L)}) = o_j(1-o_j) \\
\frac{\partial E}{\partial a_j^{(L)}} &= \frac{\partial E}{\partial o_j} = \frac{\partial \frac{1}{2}\sum_{i=1}^{k}(t_i - o_i)^2}{\partial o_j} = \frac{\partial \frac{1}{2}(t_j - o_j)^2}{\partial o_j} = -(t_j - o_j) \\
\frac{\partial E}{\partial w_{ij}^{(L)}} &= -(t_j - o_j)o_j(1-o_j)x_i^{(L-1)} \\
\delta_j^{(L)} &= \frac{\partial E}{z_j^{(L)}} = (t_j - o_j)o_j(1-o_j)) \\
\Delta w_{ij}^{(L)} &= -\eta \frac{\partial E}{\partial w_{ij}^{(L)}} = -\eta \delta_j^{(L)} x_i^{(L-1)} \\
\delta_j^{(l)} &= o_j(1-o_j)\sum_i \delta_i^{(l+1)} w_j i \\
\Delta w_{ij}^{(l)} &= -\eta \delta_j^{(l)} x_i^{(l-1)}
\end{aligned}
$$

The algorithm can also be formulated using matrices.

- $X \in \mathbb{R}^{n \times m}$ input of training data
- $T \in \mathbb{R}^{n \times k}$ target output for training data
- $W^1, \dots, W^L$ weight matrices

$$
\begin{aligned}
A^{(0)} &= A \\
Z^{(l)} &= A^{(l-1)} * W^{(1)} \\
A^{(l)} &= \phi(Z^{(l)}) \\
O &= A^{(L)} \\
E &= \frac{1}{2}\sum(T - O). * (T - O) \\
\delta^{(L)} &= (T - O). * O. * (1 - O)
\end{aligned}
\tag{2.1}
$$

2.2 Error Functions

Sometimes also called objective functions or loss-functions.

2.2.1 Mean-Squared Error

As the name suggests the mean-square error (MSE) is defined as the mean (over all training examples) of squared difference between the correct value t_i und the correct value o_i. This big errors are punished harder than small differences.

$$E_{\text{MSE}} = \frac{1}{n}\sum_{i=1}^{n}(t_i - o_i)^2 \tag{2.2}$$

This would derive to

$$\begin{aligned}\frac{\partial E_{\text{MSE}}}{\partial o_j} &= \frac{1}{n}\sum_{i=1}^{n}\frac{\partial (t_i - o_i)^2}{\partial o_j}\\ &= \frac{1}{n}\frac{\partial (t_i - o_j)^2}{\partial o_j}\\ &= \frac{1}{n}2(t_i - o_j)(-1)\\ &= -\frac{2}{n}(t_i - o_j)\end{aligned} \tag{2.3}$$

which is technically fine, but as the fraction is only a constant factor, we will often use a a slightly different definition:

$$E_{\text{MSE}} = \frac{1}{2}\sum_{i=1}^{k}(t_i - o_i)^2 \tag{2.4}$$

$$\frac{\partial E_{\text{MSE}}}{\partial o_j} = -(t_i - o_j) \tag{2.5}$$

2.2.2 Cross Entropy Error

Cross-entropy (CE) works great well on classifications tasks (t_i is either 0 or 1 and o_i is the class probability computed by the network).

$$E_{\text{CE}} = -\sum_{i=1}^{k}\left(t_i \log(o_i) + (1 - t_i)\log(1 - o_i)\right) \tag{2.6}$$

This will derive to:

$$\begin{aligned}\frac{\partial E_{\text{CE}}}{\partial o_j} &= -\frac{\partial}{\partial o_j}\sum_{i=1}^{k}\left(t_i \log(o_i) + (1 - t_i)\log(1 - o_i)\right)\\ &= -\frac{\partial}{\partial o_j}t_j\log(o_j) - \frac{\partial}{\partial o_j}(1 - t_j)\log(1 - o_j)\\ &= -\frac{t_j}{o_j} + \frac{1 - t_j}{1 - o_j}\end{aligned} \tag{2.7}$$

2.3 Activation Functions

2.3.1 Step Function

$$\phi(x) = \begin{cases} 1 & \text{if}\, x > 0 \\ 0 & \text{if}\, x \leq 0 \end{cases} \tag{2.8}$$

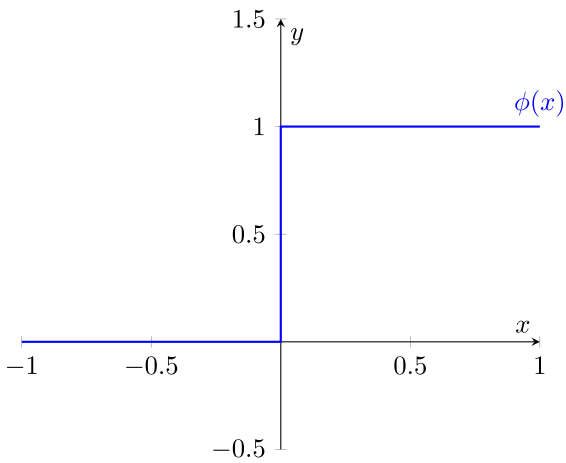

Figure 2.1: Step Function

The derivative is always 0.

2.3.2 Sigmoid

Most common activation function, can saturate and is easy to device.

$$\sigma(x) = \frac{1}{1+e^{-\beta x}} \tag{2.9}$$

And it's derivative:

$$\begin{aligned}\frac{d\sigma(x)}{dx} &= \frac{d}{dx}(1+e^{-\beta x})^{-1} \\ &= (-1)(1+e^{-\beta x})^{-2}(-\beta e^{-\beta x}) \\ &= \beta(1+e^{-\beta x})^{-2}e^{-\beta x} \\ &= \beta\frac{1}{1+e^{-\beta x}}\frac{e^{-\beta x}}{1+e^{-\beta x}} \\ &= \beta\sigma(x)\frac{e^{-\beta x}+1-1}{1+e^{-\beta x}} \\ &= \beta\sigma(x)(1-\frac{1}{1+e^{-\beta x}}) \\ &= \beta\sigma(x)(1-\sigma(x))\end{aligned} \tag{2.10}$$

2.3.3 Softmax Function

In a classification problem we would like a_j to be a probability ($\Rightarrow \sum_i a_i = 1$).

$$a_i = \phi(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{2.11}$$

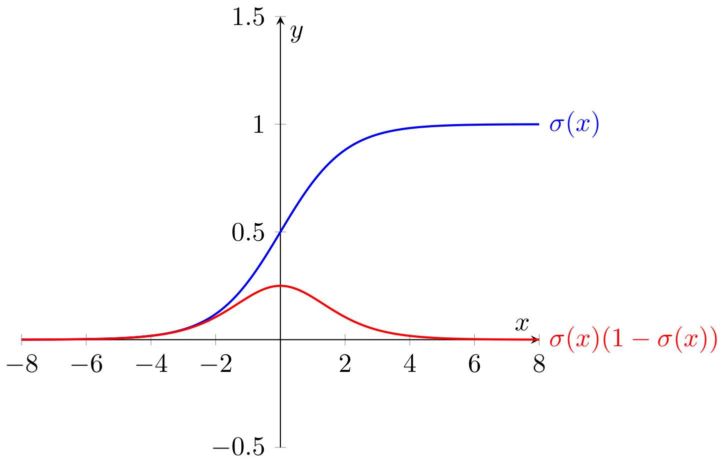

Figure 2.2: Sigmoid Function (blue) and its derivative (red).

And it's derivative:

$$
\begin{aligned}
\frac{\partial \phi(z_i)}{\partial z_i} &= \frac{\left(\frac{\partial}{\partial z_i} e^{z_i}\right)\left(\sum_j e^{z_j}\right) - e^{z_i}\left(\frac{\partial}{\partial z_i}\left(\sum_j e^{z_j}\right)\right)}{\left(\sum_j e^{y_j}\right)^2} \\
&= \frac{e^{y_i}\sum_j e^{y_j} - e^{y_i}e^{y_i}}{\left(\sum_j e^{y_j}\right)^2} \\
&= \frac{e^{y_i}\sum_j e^{y_j}}{\left(\sum_j e^{y_j}\right)^2} - \frac{e^{y_i}e^{y_i}}{\left(\sum_j e^{y_j}\right)^2} \\
&= \frac{e^{y_i}}{\sum_j e^{y_j}} - \left(\frac{e^{y_i}}{\sum_j e^{y_j}}\right)^2 \\
&= \phi(y_i) - \phi(y_i)^2 \\
&= \phi(y_i)(1 - \phi(y_i))
\end{aligned}
\tag{2.12}
$$

2.3.4 Hyperbolic Tangent Function

Similar to Sigmoid function, but if the input has a mean of 0 then so will the output.

$$
\phi(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.13}
$$

And it's derivative:

$$
\frac{d\phi(x)}{dx} = 1 - \tanh(x) = \frac{(e^x - e^{-x})^2}{(e^x + e^{-x})^2} \tag{2.14}
$$

2.3.5 Linear Function

$$
\phi(x) = x \tag{2.15}
$$

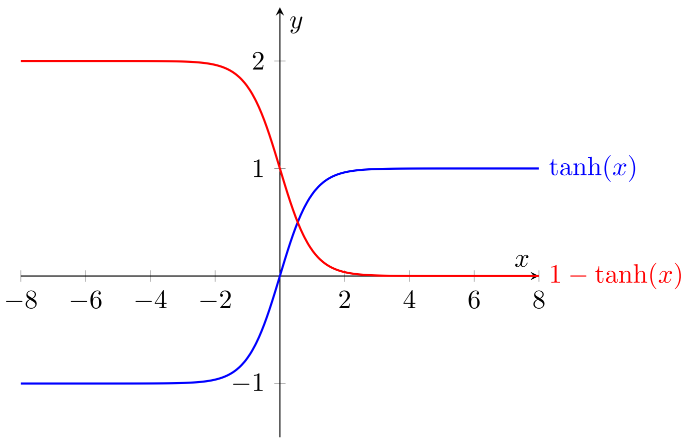

Figure 2.3: Hyperbolic Tangent Function (blue) and its derivative (red).

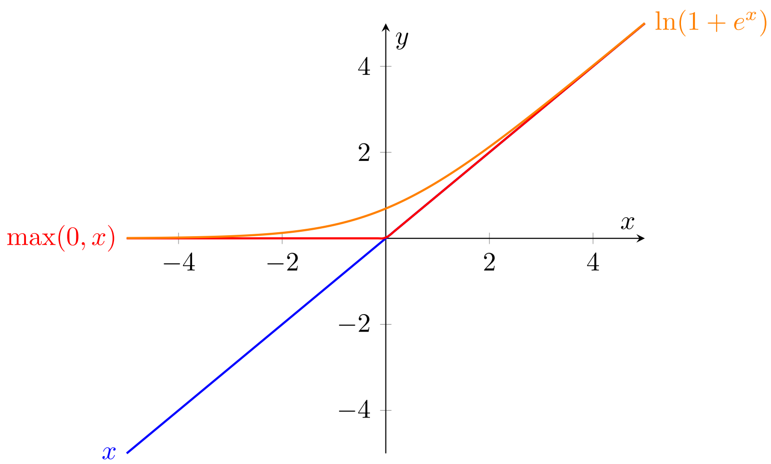

Figure 2.4: Linear Function (blue), Rectified Linear Unit (red), Softplus (orange)

2.3.6 Rectified Linear Unit

The Rectified Linear Unit (ReLU) is more biologically plausible.

$$\phi(x) = \max(0, x) \tag{2.16}$$

And it's derivative:

$$\frac{d\phi(x)}{dx} = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

2.3.7 Softplus

Smoothed version of ReLU.

$$\phi(x) = \ln(1 + e^x) \tag{2.17}$$

And it's derivative:

$$\frac{d\phi(x)}{dx} = \frac{e^x}{1 + e^x} = \frac{1}{e^{-x} + 1}$$

2.3.8 Maxout

Outputs the maximum of its inputs

$$a_j^{(l)} = \max_i z_i^{(l)}, \quad (j-1)g + 1 \leq i \leq jg \tag{2.18}$$

$$z_i^{(l)} = \sum_k x_k^{(l-1)} w_{ki}^{(l)} + b^{(l)} \tag{2.19}$$

Chapter 3

Preprocessing

3.1 Data Normalization

Numerical needs to be normalized because neural network (NN) function best with inputs between in range $[0, 1]$ or $[-1, +1]$. There are two common normalizations, the min-max normalization (sometimes called feature scaling),

$$\hat{x}_i = \frac{x_i - \min(x)}{\max(x) - \min(x)} \tag{3.1}$$

, which will transform the smallest value to 0 and the biggest to 1 and everything else linearly in between, and gaussian normalization,

$$\hat{x}_i = \frac{x_i - \mathrm{mean}(x)}{\mathrm{std}(x)} = \frac{x_i - \mu}{\sigma} \tag{3.2}$$

, will transform x to have zero mean and a standard deviation of one. Other names for int are standard score or z-scores.

Those are the two most common methods, but depending on the input there might be more.

Bibliography

[Linde et al., 1980] Linde, Y., Buzo, a., and Gray, R. (1980). An algorithm for vector quantizer design. *IEEE Transactions on Communications*, 28(1):84–95.

Acronyms

CE cross-entropy. 13

DNN deep neural network. 9, 19

KIT Karlsruhe Institute of Technology. 3

LVQ learning vector quantization. 8

MSE mean-square error. 13

NN neural network. 19